TECHNISCHE
UNIVERSITÄT
DARMSTADT

Practical Private Set Intersection Protocols for Privacy-Preserving Applications

at the Department of Computer Science
of the Technical University of Darmstadt

Doctoral Thesis

submitted in fulfillment of the requirements for the degree of
Doctor of Engineering (Dr.-Ing.)

by

Christian Weinert, M.Sc.
born in Worms, Germany

Advisors: Prof. Dr.-Ing. Thomas Schneider
Prof. Dr. Benny Pinkas

Date of submission: 02.07.2021
Date of defense: 31.08.2021

D 17
Darmstadt, 2021

Erklärung

Hiermit versichere ich, Christian Weinert, M.Sc., die vorliegende Doctoral Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, 02.07.2021

Christian Weinert, M.Sc.

Abstract

Private set intersection (PSI) protocols are cryptographic protocols that allow two parties to securely compute the intersection of their private input sets without disclosing elements outside of the intersection. While this simple functionality turns out to be instrumental for many real-world applications, existing protocol designs and implementations unfortunately incur an impractical computation and/or communication overhead. As a consequence, service providers currently deploy insecure alternatives that threaten users' privacy at a large scale.

Therefore, in this thesis, we design, implement, and evaluate practical PSI protocols to provide viable privacy-preserving alternatives for three specific application scenarios: mobile contact discovery, mutual authentication for Apple AirDrop, and database intersection analytics.

Mobile Contact Discovery. Mobile messengers commonly offer a feature that allows users to discover their existing contacts on the platform based on the phone numbers stored in their address book. Unfortunately, we find that popular messengers implement contact discovery either by uploading the users' entire address books in the clear or by using simple hashing-based protocols. As we show that such hashing-based protocols are vulnerable to brute-force attacks, the users' entire social graphs are exposed to anyone with access to the service providers' infrastructure. To instead perform the matching procedure between address books and user databases in a privacy-preserving manner, we develop and optimize two PSI protocols that are significantly more efficient than the state-of-the-art protocols of Kiss et al. (PoPETs'17) while also providing security against malicious clients.

By closely investigating the contact discovery implementation of three popular messengers (WhatsApp, Telegram, and Signal), we also find that due to insufficient rate limits, attackers can crawl the global user databases simply by enumerating phone numbers. For this problem, we propose multiple mitigation techniques, including a novel PSI-compatible rate-limiting scheme that strictly improves over the approach currently deployed by Signal.

This part of the thesis is based on the following two publications:

[KRS+19] D. Kales, C. Rechberger, T. Schneider, M. Senker, C. Weinert. **"Mobile Private Contact Discovery at Scale"**. In: *28. USENIX Security Symposium (USENIX Security'19)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2019/517. USENIX Association, 2019, pp. 1447–1464. CORE Rank A*. Appendix B.

[HWS+21] C. Hagen, C. Weinert, C. Sendner, A. Dmitrienko, T. Schneider. **"All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers"**. In: *28. Network and Distributed System Security Symposium (NDSS'21)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2020/1119. Internet Society, 2021. CORE Rank A*. Appendix A.

Mutual Authentication for Apple AirDrop. Based on the reverse-engineering efforts of Stute et al. (USENIX Security'19), we identify two privacy vulnerabilities in Apple's proprietary offline file sharing service AirDrop. They are rooted in the exchange of vulnerable hash values of contact identifiers during the authentication handshake that determines whether two device owners are mutual contacts. We demonstrate both attacks with a proof-of-concept implementation called "AirCollect" that can almost instantly recover the mobile phone numbers of nearby Apple users who open the sharing pane on their devices.

As a privacy-preserving alternative, we develop "PrivateDrop". Our solution is based on two consecutive executions of optimized PSI protocols that provide security against malicious parties and additionally enforce authentic inputs. We implement PrivateDrop in Apple's native programming language Swift and show in an empirical performance evaluation on real Apple devices that the overall authentication delay of below one second preserves the user experience of the original insecure AirDrop protocol.

This part of the thesis is based on the following two publications:

[HHS+21a] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop"**. In: *14. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/893. ACM, 2021, pp. 371–373. Appendix C.

[HHS+21b] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop"**. In: *30. USENIX Security Symposium (USENIX Security'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/481. USENIX Association, 2021, pp. 3577–3594. CORE Rank A*. Appendix D.

Database Intersection Analytics. Scenarios where two companies or governmental agencies want to conduct analyses on data subjects they have in common without revealing their entire database are usually addressed with generic circuit-based PSI protocols that can compute arbitrary functions of the database intersection. The best prior circuit-based PSI protocols of Pinkas et al. (USENIX Security'15 and ACM TOPS'18) have a complexity of $O(n \log n / \log \log n)$, where n is the number of database entries. In our work, we create the first circuit-based PSI protocol with almost linear $\omega(n)$ complexity that also significantly outperforms prior works in terms of concrete performance. Our construction is based on a novel hashing scheme called "2D Cuckoo hashing", which we analyze experimentally by spending millions of core hours on a high-performance computer.

This part of the thesis is based on the following publication:

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *37. Advances in Cryptology – EUROCRYPT'18*. Vol. 10822. LNCS. Code: https://encrypto.de/code/2DCH. Full version: https://ia.cr/2018/120. Springer, 2018, pp. 125–157. CORE Rank A*. Appendix E.

Overall, this thesis contributes to make private set intersection protocols sufficiently practical to provide privacy-preserving solutions for three widely-used real-world applications.

Zusammenfassung

Protokolle zur privaten Schnittmengenberechnung (im Englischen *private set intersection*, PSI) sind kryptographische Protokolle, die es zwei Parteien erlauben, die Schnittmenge ihrer geheimen Eingabemengen zu berechnen, ohne dass Elemente außerhalb der Schnittmenge bekannt werden. Obwohl diese einfache Funktionalität instrumental für eine Vielzahl von praktischen Anwendungen ist, erzeugen existierende Protokolle leider einen unpraktikablen Berechnungs- und Kommunikationsmehraufwand. Aus diesem Grund nutzen Dienstanbieter derzeit unsichere Alternativen, welche die Privatsphäre von Nutzern massiv gefährden.

Daher entwerfen, implementieren und evaluieren wir in dieser Thesis effiziente PSI Protokolle, um praktikable Privatsphäre-schützende Alternativen für drei spezifische Anwendungsszenarien anbieten zu können: mobile Kontaktermittlung, Authentifizierung für Apple AirDrop und Analyse von Datenbankschnittmengen.

Mobile Kontaktermittlung. Mobile Messenger bieten Nutzern in der Regel die Möglichkeit, existierende Kontakte auf der Plattform basierend auf den Telefonnummern in ihren Adressbüchern zu finden. Leider implementieren populäre Dienste diese Kontaktermittlung entweder durch das Hochladen der gesamten Adressbücher oder durch simple Hashing-basierte Protokolle. Da wir zeigen, dass solche Hashing-basierten Protokolle anfällig für Brute-Force-Angriffe sind, sind die gesamten sozialen Graphen der Nutzer für jeden einsehbar, der Zugriff auf die Infrastruktur der Dienstanbieter erlangt. Um den Abgleich zwischen Adressbüchern und Nutzerdatenbanken auf Privatsphäre-schützende Art und Weise durchzuführen, entwickeln und optimieren wir zwei PSI Protokolle, die signifikant effizienter sind als die derzeit besten Protokolle von Kiss et al. (PoPETS'17) und außerdem noch Sicherheit gegenüber böswilligen Nutzern bieten.

Durch eine nähere Untersuchung der Implementierung von Kontaktermittlungsverfahren von drei populären mobilen Messengern (WhatsApp, Telegram und Signal) finden wir außerdem heraus, dass durch unzureichende Beschränkung der Anfragen Angreifer die globalen Nutzerdatenbanken durchsuchen können, einfach indem aufeinanderfolgende Telefonnummern abgefragt werden. Für dieses Problem schlagen wir mehrere Schutzmaßnahmen vor, einschließlich eines neuen PSI-kompatiblen Verfahrens zur Beschränkung der Anfragen, das eine strikte Verbesserung gegenüber der von Signal derzeit eingesetzten Methode darstellt.

Dieser Abschnitt der Dissertation basiert auf folgenden zwei Publikationen:

[KRS+19] D. Kales, C. Rechberger, T. Schneider, M. Senker, C. Weinert. **"Mobile Private Contact Discovery at Scale"**. In: *28. USENIX Security Symposium (USENIX Security'19)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2019/517. USENIX Association, 2019, S. 1447–1464. CORE Rank A*. Appendix B.

[HWS+21] C. Hagen, C. Weinert, C. Sendner, A. Dmitrienko, T. Schneider. **"All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers"**. In: *28. Network and Distributed System Security Symposium (NDSS'21)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2020/1119. Internet Society, 2021. CORE Rank A*. Appendix A.

Authentifizierung für Apple AirDrop. Basierend auf den Reverse-Engineering-Bemühungen von Stute et al. (USENIX Security'19) identifizieren wir zwei Privatsphäre-gefährdende Schwachstellen in Apple's proprietärem Dienst AirDrop, der das lokale Teilen von Dateien ermöglicht. Diese Schwachstellen beruhen auf dem Austausch von unsicheren Hashwerten von Kontaktdaten während des Authentifizierungsschritts, der ermittelt, ob zwei Gerätebesitzer gegenseitige Kontakte sind. Wir demonstrieren beide Angriffe mit einer Machbarkeitsstudie namens "AirCollect", die fast unmittelbar die Mobilfunknummern von in der Nähe befindlichen Apple-Nutzern offenbart, die das "Teilen"-Menü auf ihren Geräten öffnen.

Als Privatsphäre-schützende Alternative entwickeln wir "PrivateDrop". Unsere Lösung basiert auf zwei aufeinanderfolgenden Ausführungen von optimierten PSI Protokollen, die Sicherheit gegenüber böswilligen Parteien gewährleisten und zusätzlich die Authentizität von Eingabewerten garantieren. Wir implementieren PrivateDrop in Apple's nativer Programmiersprache Swift und zeigen in einer empirischen Evaluation auf echten Apple Geräten, dass die Gesamtdauer des Authentifizierungsschritts von unter einer Sekunde die Benutzererfahrung des originalen, unsicheren AirDrop Protokolls aufrecht erhalten kann.

Dieser Abschnitt der Dissertation basiert auf folgenden zwei Publikationen:

[HHS+21a] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop"**. In: *14. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/893. ACM, 2021, S. 371–373. Appendix C.

[HHS+21b] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop"**. In: *30. USENIX Security Symposium (USENIX Security'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/481. USENIX Association, 2021, S. 3577–3594. CORE Rank A*. Appendix D.

Analyse von Datenbankschnittmengen. Szenarien in denen zwei Firmen oder Regierungsbehörden Analysen auf gemeinsamen Datensubjekten durchführen wollen, ohne ihre gesamten Datenbanken preiszugeben, werden üblicherweise mit generischen, Schaltkreis-basierten PSI Protokollen adressiert, die beliebige Funktionen der Datenbankschnittmenge berechnen können. Die besten vorherigen Schaltkreis-basierten PSI Protokolle von Pinkas et al. (USENIX Security'15 und ACM TOPS'18) haben eine Komplexität von $O(n \log n / \log \log n)$, wobei n die Anzahl der Datenbankeinträge ist. In unserer Arbeit entwickeln wir das erste Schaltkreis-basierte PSI Protokoll mit fast linearer $\omega(n)$ Komplexität, welches vorherige Arbeiten auch bezüglich konkreten Leistungswerten übertrifft. Unsere Konstruktion basiert auf einem neuen Hashverfahren namens "2D Cuckoo Hashing", das wir experimentell analysieren, indem wir mehrere Millionen Stunden Rechenzeit auf einem Hochleistungsrechner in Anspruch nehmen.

Dieser Abschnitt der Dissertation basiert auf folgender Publikation:

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *37. Advances in Cryptology – EUROCRYPT'18*. Bd. 10822. LNCS. Code: https://encrypto.de/code/2DCH. Full version: https://ia.cr/2018/120. Springer, 2018, S. 125–157. CORE Rank A*. Appendix E.

Insgesamt tragen wir mit dieser Thesis dazu bei, Protokolle zur privaten Schnittmengenberechnung effizient genug zu gestalten, sodass wir Privatsphäre-schützende Lösungen für drei weitverbreitete praktische Anwendungen realisieren können.

My Contributions

This thesis is based on five scientific publications that I co-authored together with my advisor Thomas Schneider and many excellent researchers as well as outstanding students: Alexandra Dmitrienko (University of Würzburg), Christoph Hagen (University of Würzburg), Alexander Heinrich, Matthias Hollick, Daniel Kales (TU Graz), Benny Pinkas (Bar-Ilan University), Christian Rechberger (TU Graz), Christoph Sendner (University of Würzburg), Matthias Senker, Milan Stute, and Udi Wieder (VMware Research). I thank all co-authors for their significant contributions to these pleasant and successful collaborations. In the following, I detail my own contributions to each publication.

Chapter 2 is based on joint works with Alexandra Dmitrienko, Christoph Hagen, Christoph Sendner, and Thomas Schneider [**HWS+21**] as well as Daniel Kales, Christian Rechberger, Thomas Schneider, and Matthias Senker [**KRS+19**].
In [**HWS+21**], I contributed to the compilation of an accurate world-wide phone number prefix database and derived masks for efficient hash reversal as well as crawling attacks. I conducted analyses on crawling results obtained by Christoph Hagen and provided a discussion on the applicability of private set intersection for mitigating hash reversal as well as crawling attacks in mobile messengers.
[**KRS+19**] is partially based on the M.Sc. thesis of Matthias Senker, which was jointly supervised by Thomas Schneider and myself, and was awarded the "Best Master Thesis Prize 2018" by "Friends of TU Darmstadt e.V.". I contributed to the survey of deployed contact discovery methods, efficiency improvements for Cuckoo filters, and the design as well as implementation of a private set intersection protocol based on the Naor-Reingold PRF. I benchmarked the server setup phase and compared all evaluation results to previous works. Together with my student assistant Oliver Schick, I integrated the implementations of [**KRS+19**] into the open-source platform CogniCrypt [**KNR+17**] to make them readily available to developers. Our submission "ContactGuard" based on [**KRS+19**] won the second prize in the "8. German IT-Security Award 2020" under my lead. I maintain the project website `https://contact-discovery.github.io` to inform the general public about recent developments related to privacy in mobile contact discovery.

Chapter 3 is based on joint works with Alexander Heinrich, Matthias Hollick, Thomas Schneider, and Milan Stute [**HHS+21a**; **HHS+21b**]. Explorative work leading to these publications was conducted by Nanako Honda as part of her M.Sc. thesis, which was jointly supervised by Matthias Hollick, Milan Stute, and myself. In [**HHS+21b**], I contributed to the design, optimization, and analysis of a private set intersection-based mutual authentication protocol for Apple AirDrop that addresses privacy vulnerabilities discovered by Milan Stute. Furthermore, I provided a Java prototype implementation that was re-implemented in Swift for our evaluation on iOS/macOS by Alexander Heinrich. For our demonstrator implementation [**HHS+21a**], I extended the rainbow table implementation of [**HWS+21**] to support SHA-256 hashing and precomputed as well as evaluated rainbow tables for German and US phone numbers. I created the project website `https://privatedrop.github.io` to inform the general public about our activities.

Chapter 4 is based on joint work with Benny Pinkas, Thomas Schneider, and Udi Wieder [**PSWW18**]. I contributed to the design of iterative 2D Cuckoo hashing, large-scale simulations for setting parameters with negligible failure probability, and the implementation as well as empirical performance evaluation of circuit-based private set intersection.

Acknowledgments

This dissertation is the outcome of spending almost exactly five years as a doctoral researcher with the Cryptography and Privacy Engineering Group (ENCRYPTO) at TU Darmstadt. The results as presented here would not have been possible without the continuous support by a variety of people. In the following, I want to thank specifically those who supported, guided, and shaped my academic journey up to now.

When I joined TU Darmstadt as an undergraduate student, I was lucky to meet a fantastic group of people even before the official start of the first semester: Michael Bugert, Marius Diebel, Simon Reuß, Lars Schulte, and Roman Uhlig. I will always cherish the fun times we had together while jointly working on numerous projects in our beloved Athene bistro.

Towards the end of my B.Sc. studies, I first came in contact with cryptography due to Johannes Buchmann's outstanding introductory course, which convinced me to write both my B.Sc. and M.Sc. thesis with his group Cryptography and Computer Algebra (CDC). I want to thank Martín Vigil for his wonderful supervision of my B.Sc. thesis. Martín also introduced me to the daily life in research after persuading me to work as his student assistant. Furthermore, I want to thank Denise Demirel who enthusiastically supervised my M.Sc. thesis and taught me all the dirty tricks in academia – knowledge that turned out tremendously helpful.

When I started as a graduate student, I was heartily welcomed by the "first generation" of ENCRYPTO PhDs: Daniel Demmler, Ágnes Kiss, and Michael Zohner. As I had almost zero prior knowledge, they patiently explained to me all the basics of secure computation and answered sheer endless questions about the ABY framework. I am also grateful to currently work with a sublime group of colleagues at ENCRYPTO: Gowri Chandran, Daniel Günther, Helen Möllering, Raine Nieminen, Oleksandr Tkachenko, Amos Treiber, and Hossein Yalame. A special shout-out to Oleksandr Tkachenko and Amos Treiber, my (virtual) office 213 partners in crime. Our daily shenanigans made also stressful times bearable.

Of course, I owe my uttermost gratitude to my advisor Thomas Schneider. During my time at ENCRYPTO, he spent significant amounts of his time and energy to provide the most excellent supervision imaginable. He was available almost literally 24/7 to instantly answer my infinite number of emails and give immediate feedback on all paper drafts as well as teaching materials. Over the time, he thereby managed to (at least partially) instill his own discipline and diligence in me. Moreover, he opened doors to many incredible opportunities, entrusted me with critical tasks, and still granted me enough freedom when necessary. Although the past years have been a remarkably intense and challenging experience, I will remain forever grateful for his support, guidance, and the provided chances to both personal as well as professional growth.

Many of my scientific publications are based on the results of dedicated B.Sc. and M.Sc. students who I had the pleasure to supervise: Susanne Felsen, Tim Heldmann, Roman Hergenreder, Nanako Honda, Tom Schuster, Matthias Senker, and Oleksandr Tkachenko. I also want to thank all of my student assistants who massively helped me by taking care of tedious

implementation tasks: Lennart Braun, Prankur Chauhan, Lukas Scheidel, Oliver Schick, and Oleksandr Tkachenko (yes, the same Oleksandr I thanked already three times above).

Throughout the past years, I have been supported by various team assistants, whom I would like to thank for taking care of all annoying paperwork and handling my countless reimbursement requests: Karina Köhres, Heike Meißner, Melanie Schöyen, and Petra Fuhrmann. Furthermore, I would like to thank our lab technician Jens Adler, especially for somehow procuring my precious keyboard, a "solid yet very basic model". On the administrative side, I also want to thank Johannes Braun and Stefanie Kettler, who supported me as part of the collaborative research center CROSSING. The bi-weekly CROSSING research seminars and the bi-annual CROSSING retreats were always highlights to look forward to.

The papers included in this thesis received a fair amount of press coverage and were awarded several prices. This was only possible due to the efforts of Ann-Kathrin Braun, Elena Dröge, Daniela Fleckenstein, and Cornelia Reitz. They supported all our PR activities and force-fed private set intersection research to the general public on our behalf.

The final step to complete the PhD process was of course a successful defense. I am incredibly grateful to have had Benny Pinkas as my external advisor as well as Marc Fischlin, Matthias Hollick, and Iryna Gurevych as part of my PhD committee. It was an honor to defend in front of such a highly distinguished committee and I appreciated the interesting discussion.

Last but not least, I want to thank my parents and my extended family for generously supporting all my studies and keeping me well-nourished during the weekends. Assuming I will stay in academia, I will unfortunately never be able to pay anything back.

Contents

1 Introduction

One of the ongoing global mega trends is digitization. This trend is further accelerated with the COVID-19 pandemic forcing people world-wide to move their entire professional and to large extents also personal lives to online platforms [SC20]. Consequently, the amount of sensitive data that is produced daily in digital form and shared with the platform providers that power this digital revolution increases significantly [Dom20]. The responsible utilization and protection of such sensitive data is a strict requirement to enable trust and sustainable growth in digital ecosystems [Gla17]. Despite regulatory efforts to enforce data protection and data minimization principles (e.g., with the European GDPR [Eur16]), we unfortunately continue to witness a staggering number of reports of data breaches [Hun19] and abuse [Con18]. In April 2021 alone, personal contact information (including phone numbers and email addresses) of more than 1 billion users were leaked from major platform providers such as Facebook [Hol21] and LinkedIn [Can21a].

To some extent, these incidents simply stem from lacking protective measures, but also the excessive collection, processing, and exchange of sensitive user data to facilitate convenience features [New21] and/or to increase the platform provider's profit [WE21]. While some platform providers lobby to establish the narrative that certain types of data leaks are a completely normal phenomenon [Can21b; Ham21], we hope and show there is a better way. Specifically, cryptographic techniques from the area of secure computation make it possible to utilize sensitive data while at the same time providing strict and well-defined guarantees in terms of protecting confidentiality. One of these techniques that we closely investigate in the following is so-called Private Set Intersection (PSI).

Set intersection itself is a trivial mathematical operation that computes $Z = X \cap Y$ for two input sets X and Y. However, when two dedicated parties, Alice and Bob, each have one set and want to compute their intersection, things get more interesting. Of course, Alice could simply give all her data to Bob for him to compute the intersection and return the result. Unfortunately, this gives Bob access to Alice's entire private and potentially sensitive data set. Instead, we want to realize the ideal functionality depicted in Figure 1.1, where a trusted third party receives the input sets over secure channels and performs the operation on behalf of Alice and Bob. In the end, the trusted third party only discloses the elements in Z but no information about elements in $(X \cup Y) \setminus Z$, i.e., elements outside of the intersection remain confidential. This is what we refer to with the term Private Set Intersection (PSI).

Unfortunately, technical equivalents of trusted third parties do not exist in reality. The closest approximation are Trusted Execution Environments (TEEs) such as Intel Software Guard Extensions (SGX) [Int19] that shield sensitive code and data in so-called enclaves

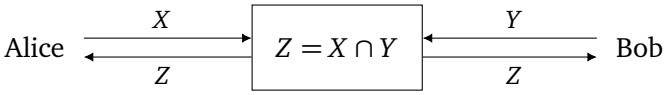


Figure 1.1: Ideal functionality for Private Set Intersection (PSI) between Alice and Bob on their respective private input sets X and Y. Depending on the use case, output Z can be returned to both or only one of the parties.

via hardware isolation and cryptographic techniques to guarantee confidentiality as well as integrity of execution. However, TEEs have been successfully compromised time and again, especially via subtle side-channel attacks (e.g., [BMW+18; BMS+20]). Nevertheless, it is possible to realize the ideal functionality by emulating the behavior of the trusted third party in an interactive cryptographic protocol that is proven to be secure in presence of certain types of adversaries. Here, we mainly distinguish between semi-honest and malicious adversaries. Semi-honest adversaries in an "honest-but-curious" fashion try to learn as much information as possible while still adhering to the protocol specification (e.g., because they are incentivised by the threat of financial penalties). On the contrary, malicious adversaries might even deviate arbitrarily from the protocol in an attempt to cheat.

Cryptographic two-party PSI protocols exist since the 1980's [Sha80; Mea86], but are still a very active and surprisingly broad field of research. As a sub-field of cryptography under the umbrella term of secure computation, the landscape of two-party PSI protocols can be categorized according to numerous aspects. In addition to the mentioned adversary models, an important distinction is to be made between *specialized* and *generic* or so-called *circuit-based* PSI. Specialized PSI protocols rely on cryptographic building blocks such as Diffie-Hellman key exchange, blind-RSA, El-Gamal encryption, Homomorphic Encryption (HE), Oblivious Transfer (OT), or Oblivious Pseudo-Random Functions (OPRFs) to securely compute nothing but the intersection itself [FNP04; JL09a; CKT10; CT10; JL10; BBC+11; CT12; DCW13; PSZ14; PSSZ15; FHNP16; KKRT16; CLR17; KLS+17; RR17a; RR17b; CHLR18; PSZ18; RA18; **KRS+19**; PRTY19; PRTY20]. On the other hand, generic PSI protocols utilize Multi-Party Computation (MPC) protocols such as Yao's garbled circuits [Yao82; Yao86] or the protocol by Goldreich, Micali, and Wigderson (GMW) [GMW87] that can securely evaluate Boolean circuits to determine the intersection [HEK12; PSZ14; PSSZ15; **PSWW18**; PSZ18; FNO19; PSTY19; KK20; CGS21; RS21]. Besides computing the intersection, this approach allows them to trivially compute arbitrary functions on top of the intersection that might be of interest – without disclosing the intermediate intersection result. However, the lines are blurred, as there also exist specialized protocols to compute specific PSI variants (e.g., PSI-Sum [IKN+17; IKN+20; MPR+20] and PSI-Cardinality [CGT12; DD15; EFG+15]) as well as hybrid constructions [CO18; GMR+21]. We visualize this simplified categorization in Figure 1.2.

The reason for all these ongoing research activities and directions is that early developments as well as completely generic approaches turned out impractical in terms of required computation (how much time is spend on cryptographic operations) and communication (how much data must be transferred between the parties) when being applied to real-world applications

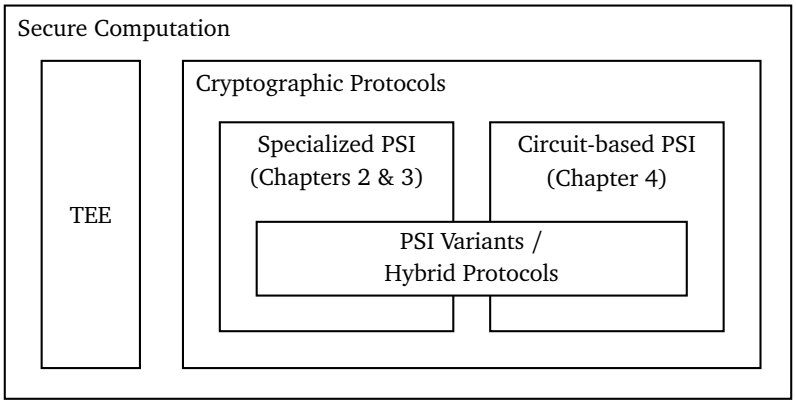


Figure 1.2: Simplified landscape of PSI protocols.

and sets with millions or even billions of elements. Despite the many applications for PSI that have been proposed theoretically over the years, there is a significant gap to truly practical solutions for concrete use case scenarios. Only in recent years, we see first deployments of PSI, e.g., Google running a PSI variant with their customers to securely compute the conversion rate for online advertisements [IKN+20] and an integration into Google's Chrome browser for compromised credential checking [TPY+19].

The goal of this thesis is to significantly advance the practicality of specialized as well as generic PSI protocols with concrete real-world applications in mind. Ultimately, we hope to facilitate further large-scale deployments of PSI by reaching new levels of practical performance. For this, we investigate three applications that at their core revolve around the set intersection problem. For these applications, we demonstrate the shortcomings of currently deployed insecure and privacy-invasive systems by devising attacks to exploit inherent vulnerabilities that leak sensitive user data. Then, we design, optimize, implement, and evaluate practical PSI protocols to provide alternative privacy-preserving solutions. Furthermore, we engage in the public discourse by responsibly disclosing the privacy vulnerabilities that we discovered to service providers, raising awareness for existing privacy risks and our alternative privacy-preserving solutions with media coverage, and presenting our research results to the general public in accessible formats.

In the following, we first introduce each of the studied applications on a high level. Then, we outline our contributions towards investigating and making these applications privacy-preserving via PSI.

1.1 Applications for Private Set Intersection

Set intersection as depicted in Figure 1.1 is a very basic functionality. However, it turns out to be instrumental for a variety of real-world applications. In this thesis, we closely study three applications in three different settings. In a business-to-consumer setting, we

study mobile contact discovery, in a consumer-to-consumer setting the mutual authentication used in Apple's AirDrop protocol, and in a business-to-business setting analytics for database intersection. In the following, we briefly describe each setting, discuss the applicability of set intersection, and the associated (privacy) challenges.

Mobile Contact Discovery. Mobile messaging applications like WhatsApp[1], Telegram[2], and Signal[3] all provide a feature that allows users to immediately start messaging existing contacts from their address book that are also registered with the respective messenger. This feature is called mobile contact discovery. It utilizes the fact that users are identified via their mobile phone numbers to match address book entries against the database of registered users and thereby discover existing contacts. As visualized in Figure 1.3, this can be modeled as a set intersection problem by having the clients input their address book and the service providers their user database to the ideal functionality.

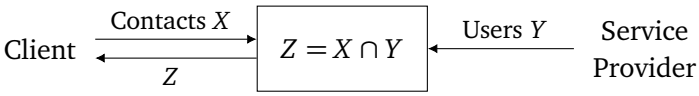


Figure 1.3: Ideal functionality for PSI when applied to mobile contact discovery between a client with contact set X and a service provider with user database Y.

PSI for mobile contact discovery, which we also call *mobile private contact discovery*, enables to conduct this process in a privacy-preserving way as users only have to reveal those of their contacts that are actually registered with the messengers. Also, the database of the service provider is protected.

Proposing a suitable PSI protocol for mobile private contact discovery comes with a number of challenges. First, there are strict performance requirements as users expect almost immediate results and also efficiency requirements due to the resource-constraint mobile setting. Second, one has to accommodate especially for malicious clients that can arbitrarily manipulate their application code (without facing penalties) in an attempt to extract the service provider's global user database.

Mutual Authentication for Apple AirDrop. AirDrop is Apple's proprietary wireless protocol that allows users to conveniently transfer files between nearby Apple devices, e.g., iPhones, iPads, or MacBooks. Whenever Apple users open the sharing pane in iOS or macOS, AirDrop automatically scans for potential receiver devices and displays the results including contact names and pictures. Furthermore, AirDrop offers a so-called "contacts only" mode. Receivers with this mode enabled answer incoming requests only from known contacts. To map nearby devices to contacts and to implement the "contacts only" mode, the AirDrop protocol generally performs an authentication handshake where the involved devices check whether they are mutual contacts. This authentication phase involves checking if one of the sender's own

[1] https://whatsapp.com
[2] https://telegram.org
[3] https://signal.org

phone numbers or email addresses is stored in the receiver's address book, and vice versa. This semantic as implemented in AirDrop can be modeled as two consecutive instances of set intersection as visualized in Figure 1.4[4].

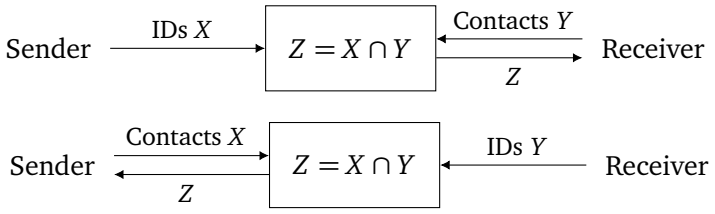


Figure 1.4: Ideal functionality for PSI when applied to mutual authentication for Apple AirDrop. IDs is a set of the users' own phone numbers and email addresses, whereas contacts is a set of phone numbers and email addresses stored in the users' address books.

PSI for AirDrop's mutual authentication protocol allows both parties to keep their contact identifiers and address book entries completely private in case no match is found. In case a match is found, only the matching identifiers are revealed. As for mobile contact discovery, we must consider user expectations regarding the resulting authentication delay, the resource constraints of mobile devices, and malicious behavior of users.

Database Intersection Analytics. Companies and government agencies often face use cases where performing analyses over commonly known subjects (e.g., customers or citizens) would yield (mutual) benefits. For example, two companies could be interested to know how many customers they have in common in order to decide whether initiating a joint marketing campaign is a promising idea. To enable such analyses, it is common practice to exchange or sell entire databases – unless the data is so sensitive that such endeavors are prohibited by law (e.g., medical data). This process, which we call database intersection analytics, can be also modeled as a generic case of set intersection as visualized in Figure 1.5. Here, a function f is applied on the intersection result to output the desired analysis instead of the plain intersection. For more advanced analyses, it is also possible to perform analytics over data associated with the primary keys that determine the intersection.

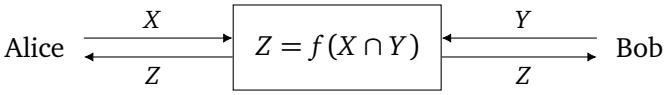


Figure 1.5: Ideal functionality for PSI when applied to database intersection analytics between Alice and Bob on their respective private input sets X and Y with function f to be computed on the intersection result. Z not necessarily is a set of elements but represents an arbitrary computation result in the range of f. Elements in X and Y may contain additional payloads to be considered by f.

[4]Alternatively, it could also be modeled with a variant of set intersection that returns a single Boolean value indicating whether the intersection is empty or not.

PSI for database intersection analytics allows both parties to keep unrelated database entries private and not even leak the intersection result. Instead, only the output of the agreed-upon analysis function is revealed, which might be only a single bit indicating whether the intersection is empty or not. For this, the function must be symmetric, i.e., the result must be independent of the order of inputs. In terms of requirements, it is often possible to argue that security against semi-honest adversaries is sufficient. This is because in a business-to-business context, the involved parties may face severe legal, financial, and reputational risks when being caught cheating.

1.2 Thesis Outline and Summary of Contributions

The following chapters describe our contributions to make each application presented in §1.1 privacy-preserving via practical PSI protocols. Furthermore, we describe the impact of our research and put it into context of related work and recent developments.

We briefly summarize the chapters as follows:

Chapter 2 We first investigate how mobile contact discovery is currently implemented in popular mobile messengers [**KRS+19**; **HWS+21**]. Unfortunately, all messengers either directly upload all phone numbers stored in the users' address books to the service providers or run a naive hashing-based protocol. As we demonstrate with new optimized techniques, curious or compromised service providers can reverse received hashes of mobile phone numbers within milliseconds [**HWS+21**]. As an alternative, we propose two significantly optimized two-party PSI protocols based on [PSSW09; HL10; KLS+17] with security against malicious clients [**KRS+19**]. Compared to previous works [CLR17; KLS+17; CHLR18; RA18], we reach practical performance for large-scale user databases as our empirical evaluation on Android devices over Wi-Fi and LTE connections demonstrates. We make our private contact discovery implementations readily available to application developers via the open-source platform CogniCrypt [**KNR+17**] for secure integration of cryptographic software.

Furthermore, in [**HWS+21**] we find that current deployments of mobile contact discovery severely threaten the privacy of users as major services can be crawled at large scale via enumeration attacks due to insufficient rate limits. Specifically, we conduct large-scale crawling attacks on WhatsApp, Telegram, and Signal and perform analyses on the gathered statistics to provide unique insights into the privacy awareness of users. As a mitigation for such attacks, we design and evaluate a PSI-compatible rate-limiting scheme for services like Signal that do not store a synchronization state for contact discovery on their server.

We maintain a project website at `https://contact-discovery.github.io` with comprehensible explanations of our research in English and German. Additionally, an "explainer video" is available in English and German on YouTube at `https://youtu.`

be/4vgKHmNaAAw and https://youtu.be/P_K166jvG7U, respectively. Our press releases received significant national as well as international media coverage as documented at https://encrypto.de/news/contact-discovery. Moreover, our work in [**KRS+19**] received the second prize in the German IT-Security Award 2020 as an outstanding IT security innovation of great practical and market relevance.

Chapter 3 In our work [**HHS+21b**], we first investigate Apple's current AirDrop implementation. It turns out that the mutual authentication protocol is implemented by exchanging hash values of contact identifiers (phone numbers and email addresses), which are vulnerable to various brute-force and dictionary attacks due to their low entropy. We describe two practical attack scenarios and provide a proof-of-concept implementation that can extract mobile phone numbers of Apple users within seconds.

As a remediation, we propose "PrivateDrop" utilizing two consecutive executions of a maliciously secure two-party PSI protocol based on [JL10] and enriched with signed inputs [CZ09; CKT10; CT10]. We suggest several optimizations to improve performance and integrate PSI tightly into the AirDrop protocol flow. Our prototype implementation evaluated on actual iPhones and MacBooks demonstrates the practicality of our approach with an authentication delay well below one second.

We maintain a project website at https://privatedrop.github.io with comprehensible explanations of our research in English and German. Our press releases received national and especially international media coverage as documented at https://encrypto.de/news/privatedrop.

Chapter 4 We present the first two-party semi-honest circuit-based PSI protocol for privacy-preserving database intersection analytics with $\omega(n)$ complexity, where n is the number of elements in the input set of each party [**PSWW18**]. This almost linear complexity is reached by designing a novel hashing scheme that we call "2D Cuckoo hashing". Since the iterative version of this algorithm is too complicated to be analyzed formally, we accurately determine its failure probability via large-scale experiments and set parameters accordingly. These experiments required 5.5 million core hours on the Lichtenberg high-performance computer of the TU Darmstadt.

Our work made first important steps towards practical circuit-based PSI with linear complexity. Following our research, there have been more recent works with $O(n)$ complexity and concrete performance improvements [PSTY19; CGS21; RS21] as well as other constructions for database intersection analytics [BKM+20; LPR+20; MRR20; GMR+21], which we review in this chapter.

The final Chapter 5 concludes this thesis with suggestions for future work.

1.3 Open Access and Responsible Disclosure

The full versions of all papers provided in the appendices of this dissertation are freely available on the IACR Cryptology ePrint Archive at https://eprint.iacr.org. The source code of all prototype and proof-of-concept implementations described in this thesis, which were also used for our empirical performance evaluations, is publicly available on GitHub at https://github.com/encryptogroup and further repositories clearly linked in the respective publications. This helps other researchers to reproduce our results and build upon our software. By choosing the software licenses as permissive as possible (e.g., MIT License), we hope our code is even used by companies as inspiration for creating production-ready implementations of privacy-preserving applications.

We reported the privacy vulnerabilities discovered in the course of our research via responsible disclosure procedures to Apple, Facebook, Signal, and Telegram, and coordinated with the companies the timelines of our publications. For our findings, Facebook awarded a bug bounty, which we donated to the Electronic Frontier Foundation (EFF) in order to support their activities in defending fundamental privacy rights.

2 Privacy-Preserving Mobile Contact Discovery

Mobile contact discovery refers to a feature implemented in most mobile messengers (e.g., WhatsApp[1], Telegram[2], and Signal[3]) that allows users of such applications to conveniently connect with their existing contacts. For this, all contacts stored in the address book application of a user's phone are checked against the database of registered users of a messenger. As phone numbers serve as the primary identifier, users can therefore immediately start messaging existing contacts that are also registered with a particular service. This procedure initially happens when launching a newly installed messenger for the first time, which is why such applications usually request permission to access the address book. Furthermore, a synchronization is triggered whenever new contacts are added to the address book and periodically to update the status of existing contacts.

Since address book entries accurately represent a user's social graph, such data must be considered as sensitive information and should be protected accordingly. In contrast to social graphs visible on social network platforms such as Facebook, address books additionally contain more sensitive contacts. For example, which kind of doctors a person is seeing enables conclusions from which disease someone is suffering. Such knowledge could in turn be abused for discrimination (e.g., employers refusing to hire someone due to expected sick leaves) as well as for targeted advertisement or scams (e.g., ineffective yet expensive treatments for desperate patients). Furthermore, knowledge about secret contact relationships (e.g., a whistleblower being in contact with a journalist) could be abused to blackmail users with the threat to harm their reputation by publishing the information or by making them the target of an investigation.

Likewise, the databases of service providers are an attractive target that should be protected sufficiently. This is because the simple information which phone numbers are actively using a particular messenger can be valuable to attackers. For example, attackers could utilize such databases to compromise a large number of devices with ransomware once they find a security vulnerability in a messaging application that can be exploited by sending a specifically crafted message. Moreover, such databases usually not only contain the binary information whether a particular number is registered, but also further sensitive profile data that is revealed during the contact discovery process. User profiles often include profile picture(s), short status texts, and timestamps when the user was last seen online. Such data can be abused to build detailed

[1] https://whatsapp.com
[2] https://telegram.org
[3] https://signal.org

user profiles when enriched with other data sources, and to accurately monitor user behavior via timestamps.

Unfortunately, it is rather unclear how mobile contact discovery is implemented in practice, especially which measures are in place to protect users' social graphs as well as service providers' databases. This to some extent even applies to open-source messengers like Signal with reports that the deployed server code diverges from what is available in the public repository, which has not been updated for over a year [Böc21].

In this chapter, we therefore describe our contributions in terms of investigating how contact discovery is currently implemented for popular messengers (cf. §2.1.1). As we find that neither users' social graphs nor the service providers' databases are properly protected, we propose and evaluate secure alternatives based on Private Set Intersection (PSI, cf. §2.1.2) that scale to large user databases. Finally, we put our efforts into perspective by comparing them to relevant related works (cf. §2.2).

2.1 Our Contributions

Our contributions towards privacy-preserving mobile contact discovery are two-fold: First, we convincingly demonstrate that currently deployed contact discovery methods are vulnerable to large-scale abuse (cf. §2.1.1). Then, as a replacement for currently deployed insecure methods, we propose scalable PSI protocols that are specifically optimized for the use case of mobile private contact discovery (cf. §2.1.2).

2.1.1 Large-scale Abuse of Contact Discovery in Mobile Messengers

In a preliminary study contained in [**KRS+19**] (cf. Appendix B), we find by analyzing privacy policies, network traffic, and source code that contact discovery in common mobile messengers is implemented by either regularly uploading all phone numbers stored in the users' address books to the service providers or by using a naive hashing-based protocol[4], where phone numbers are transferred in hashed form to the service providers. Uploading all phone numbers directly leaks the users' social graphs to the service providers, where they are even stored to be able to push immediate contact discovery updates when one of the contacts joins the service. Also, for the naive hashing-based protocol it is known that brute-force attacks can reveal the original phone numbers due to the low entropy of the hashed contact identifiers that are transferred to the service providers [Mar14; DKCL18; MZM+18].

[4] A notable exception is Signal with the introduction of an Intel SGX-based contact discovery service [Mar17], which however has many other shortcomings as discussed in §2.2.2.

This leaves two initial questions:

1. How trivial is it for curious service providers, governmental agencies, or attackers compromising the service provider to reconstruct the users' entire address books when obtaining the hashed phone numbers?
2. Are at least the service providers' databases properly protected from malicious users trying to illegitimately access a large number of profiles via the contact discovery API?

This thesis extensively answers both questions with the following publication that can be found in Appendix A:

[HWS+21] C. HAGEN, C. WEINERT, C. SENDNER, A. DMITRIENKO, T. SCHNEIDER. **"All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers"**. In: *28. Network and Distributed System Security Symposium (NDSS'21)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2020/1119. Internet Society, 2021. CORE Rank A*. Appendix A.

Concretely, we show that reversing hashes of mobile phone numbers is possible in the order of milliseconds, even for attackers with consumer hardware. Moreover, we demonstrate for three popular messengers (WhatsApp, Telegram, and Signal) that large-scale abuse of the contact discovery API is possible, i.e., malicious users can excessively query for profiles via crawling attacks due to insufficient rate limits and the lack of other protective measures. In the following, we provide a more detailed summary of our core findings.

Mobile Phone Number Prefix Database. In order to accurately quantify the difficulty of both brute-force as well as crawling attacks on contact discovery, it is instrumental to have an accurate estimation for the size of the phone number space. For this, we compile a world-wide database of phone number prefixes, including the length of the corresponding subscriber numbers, mainly from data provided by the International Telecommunication Union (ITU)[5]. In a second step, we filter out number blocks that are rejected for various reasons by Google's validation tool `libphonenumber`[6] and the WhatsApp registration/login API. Since users primarily have mobile numbers registered with mobile messengers, we can further reduce the search space by excluding land-line numbers. As a result, we can state that currently there are about 118 billion registrable mobile phone numbers world-wide and about 0.5 billion in the US. These numbers equal only about 36.78 bit and 28.91 bit of entropy, respectively. Interestingly, we find that the size of the search space vastly differs between countries. For example, Austria has almost 10 k registrable mobile phone numbers per citizen, making attacks much more difficult than for the US with a ratio of 1.5.

[5] https://www.itu.int/oth/T0202.aspx?parent=T0202
[6] https://github.com/google/libphonenumber

Hash Reversal Attacks (Question 1). The low entropy of mobile phone numbers already hints at the feasibility of hash reversal attacks. To exactly quantify the difficulty, we experiment with three approaches: large-scale look-up databases, brute-force attacks, and a novel rainbow table construction. For instantaneous look-ups, we set up a cluster of Redis in-memory databases[7] that consume 630 GB of RAM per 8 billion numbers, but can reverse batches of 10 k SHA-1[8] hashes in 1 s. Using brute-force attacks on a GPU with the hybrid mode of hashcat[9], we can reverse any mobile phone number hash within at most 16 h, resulting in an amortized time of 57 ms for batches of 1 million hashes. Since the first two approaches are either resource-intensive or perform only well for large batches, we also study rainbow tables [Hel80; Oec03]. As existing rainbow table implementations for password cracking are extremely inefficient for the purpose, we design our own reduction function that can deal with the prefix-dependent non-uniform input domain of mobile phone numbers. Our implementation called "RainbowPhones" based on RainbowCrack[10] requires 24 GB of storage for rainbow tables that can reverse batches of 10 k mobile phone number hashes with 99.99 % success rate in 8.67 min (52 ms amortized). As we thus conclude that hashing-based contact discovery provides no protection for users' social graphs whatsoever, we discuss the use of PSI as a viable alternative in §2.1.2.

Crawling Attacks (Question 2). We investigate the protection measures of the contact discovery API for three popular mobile messengers (WhatsApp, Telegram, and Signal) by imitating the behavior of malicious users that try to crawl user databases via enumeration attacks, i.e., requesting the registration state and potentially additional profile information for each possible mobile phone number. For this, we use very little resources: one laptop to run Android emulators (WhatsApp) and rudimentary self-developed clients (Telegram and Signal), a VPN subscription to rotate IP addresses, the free Hushed application[11] to register accounts with temporary numbers, and about one month time for all experiments.

For WhatsApp, we crawled 10 % of all mobile phone numbers in the US and found that it allows users to synchronize about 60 k contacts per day, whereas there seems to be no upper limit on the total amount of contacts one account can query: one of our accounts crawled 2.86 million numbers without any ramifications. Due to the lax default privacy settings, one can additionally access a public profile picture, status text, and the last online timestamp. We find public profile pictures and status texts with about 49.6 % and 89.7 % of users, respectively.

Telegram has surprisingly strict rate limits: after adding 5 k numbers in total, only additional 100 numbers can be added per day. However, Telegram exposes more profile information than WhatsApp (up to 100 public profile pictures per user) and even information about numbers not registered with the service: the so-called "importer count" states for each

[7] https://redis.io

[8] We choose SHA-1 as the hash function in our studies as truncated SHA-1 hashes are used by Signal for performing hashing-based contact discovery.

[9] https://hashcat.net

[10] https://github.com/inAudible-NG/RainbowCrack-NG

[11] https://hushed.com

non-registered number how many Telegram users have stored this number as a contact. The purpose of this importer count is to make clever suggestions to invite friends that would supposedly benefit a lot from using Telegram due to a large existing network.

Signal in contrast to WhatsApp and Telegram does not store users' contacts on the server. Thus, users must be able to query their entire address books in each request. We find that each account can query 50 k contacts four times a day. These generous rate limits allowed us to crawl 100 % of *all* mobile phone numbers in the US. Since mobile phone numbers in the US can be mapped to the individual states, we can make interesting observations, e.g., numbers in Washington D.C. are more than twice as likely to be registered with Signal than in any other state. While Signal exposes no public profile information, we find that 46.3 % of Signal users that are also registered on WhatsApp have a public profile picture on WhatsApp. This to some extent contradicts the expectation of Signal users being especially privacy-aware.

Incremental Contact Discovery. We discuss a wide range of mostly known techniques to detect or even prevent crawling attacks (e.g., honeypot numbers to detct enumerations or CAPTCHAs to validate suspicious requests). Additionally, we propose a new contact discovery scheme called *incremental contact discovery* for services like Signal that do not store user data on the server side. In this scheme, the server of the service provider maintains two sets of users: one for the entire user database and one for users registered or unregistered within a certain time interval. Clients are then restricted to perform only one initial synchronization with the full set but can perform regular syncs with the sparsely populated set containing the incremental changes. The improvement over Signal's current approach depends on the relative number of changes to the full set. Assuming that the change rate is between 1.0 % and 0.01 % per day, crawling attackers can discover 10× to 100× less users per day, respectively. Our scheme is a strict improvement over Signal's current approach, has virtually no overhead, is easy to implement, and compatible with PSI, which can be used on top to securely sync with both sets.

Impact. We reported our findings to the respective service providers via their standard responsible disclosure routines. As a result, WhatsApp and Signal adjusted their rate limits and implemented further measures to increase the difficulty for attackers to conduct large-scale crawling attacks. Potentially related, Signal plans to add support for alternative identifiers, which results in a significantly larger search space, and for messaging contacts that are not stored in the address book, which prevents leaking sensitive contacts to other third-party applications [Sig20]. For our findings, Facebook (WhatsApp's parent company) awarded us a bug bounty, which we donated to the Electronic Frontier Foundation (EFF). Due to its implications for the privacy of billions of users, our work received significant media coverage, which is documented at `https://encrypto.de/news/contact-discovery`.

2.1.2 Mobile Private Contact Discovery at Scale

Currently deployed contact discovery methods in popular mobile messengers either directly or indirectly leak users' entire social graphs by uploading all contact identifiers of address

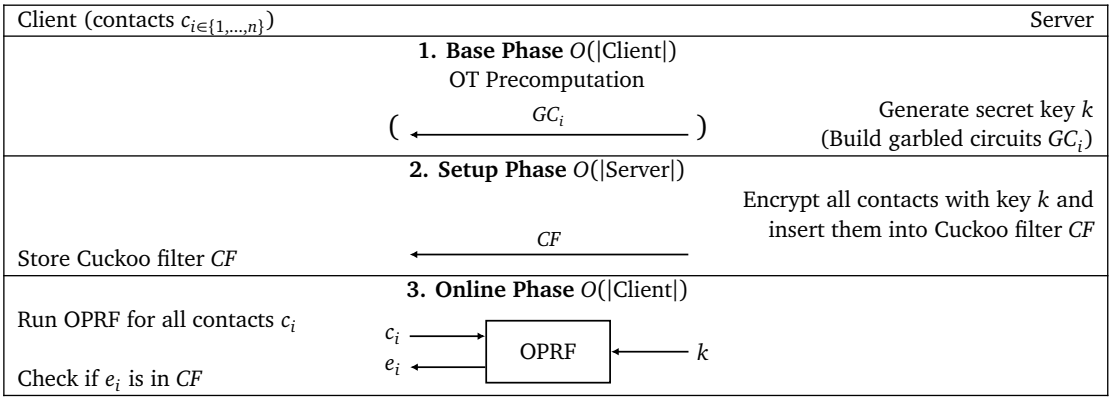


Figure 2.1: Protocol phases for Oblivious Pseudo-Random Function (OPRF)-based unbalanced PSI in precomputation form [KLS+17] instantiated with Cuckoo filters [FAKM14; RA18].

book entries in the clear or hashed form (cf. §2.1.1). This can be prevented by using Private Set Intersection (PSI) protocols to securely compute the intersection between address book entries and the database of registered users. As a result, the service provider (and anyone monitoring or compromising the service provider's infrastructure) can only learn contact identifiers that are already registered with the service.

There specifically exists a class of *unbalanced* PSI protocols that are optimized for the use case where one set (the user's address book) is much smaller than the other set (the database of the service provider). More precisely, unbalanced PSI protocols have an online communication complexity that is linear in the size of the smaller set, and shift the major communication complexity to a setup phase that can be run at an arbitrary point in time before the actual computation (e.g., overnight when the phone is charging and connected to Wi-Fi) and usually is a one-time cost. The protocol flow in this precomputation form as introduced in [KLS+17] is depicted in Figure 2.1 and the individual phases are briefly described in the following:

Base Phase In the base phase, Oblivious Transfer (OT) precomputation takes place and the server of the service provider generates a secret key k. Depending on the exact protocol to be executed in later phases, also Yao's garbled circuits [Yao82; Yao86] are precomputed and sent to the client. The computation and communication complexity of this phase is linear in the size of the client's input set, i.e., the address book.

Setup Phase In the setup phase, the server encrypts all database entries with the secret key k and inserts them in a probabilistic data structure for efficient membership testing. In our case we instantiate this data structure with a so-called Cuckoo filter [FAKM14; RA18]. This Cuckoo filter is then transferred to and stored by the client.

Online Phase Finally, in the online phase, client and server engage in an Oblivious Pseudo-Random Function (OPRF) evaluation, where the client obliviously obtains encryptions

of its address book entries under the server's secret key k. The client can then test this encryption against the stored Cuckoo filter.

Unfortunately, existing unbalanced PSI protocols and implementations [CLR17; KLS+17; CHLR18; RA18] turn out to be impractical when used at scale (i.e., considering realistic database sizes with millions or even billions of entries). Also, malicious behavior is not considered [RA18] but a critical aspect because users can arbitrarily modify the client code running on their phones in an attempt to extract the service provider's confidential user database.

This thesis has contributed significantly to develop unbalanced PSI protocols for mobile contact discovery at scale that are secure against malicious clients with the following publication that can be found in Appendix B:

[KRS+19] D. KALES, C. RECHBERGER, T. SCHNEIDER, M. SENKER, C. WEINERT. **"Mobile Private Contact Discovery at Scale"**. In: *28. USENIX Security Symposium (USENIX Security'19)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2019/517. USENIX Association, 2019, pp. 1447–1464. CORE Rank A*. Appendix B.

We propose two unbalanced PSI protocols: one is based on the oblivious evaluation of a block cipher in circuit representation [PSSW09; KLS+17] (GC-PSI), the other one on the Naor-Reingold PRF [HL10; KLS+17] (NR-PSI). For both protocols, we optimize each component in the protocol flow depicted in Figure 2.1, provide native implementations that utilize capabilities of modern smartphone Systems-on-a-Chip (SoCs), and conduct performance evaluations in realistic settings. Furthermore, we make our implementations readily available to developers via the open-source platform CogniCrypt [**KNR+17**] such that even non-experts can securely integrate our code into their applications. For settings where the availability of non-colluding servers can be assumed, we additionally propose an extension that combines our PSI protocols with Private Information Retrieval (PIR) to further reduce the required communication overhead.

More Efficient Cuckoo Filters. For efficiently distributing the service provider's encrypted database, unbalanced PSI protocols initially considered Bloom filters [Blo70] as a suitable probabilistic data structure to reduce the required amount of communication. Later in [RA18], so-called Cuckoo filters were suggested as a drop-in replacement, which are easier to maintain, provide faster look-ups, and most importantly have a better storage space efficiency [FAKM14]. Since previous works set rather unrealistic parameters (e.g., a false positive probability of 2^{-13} [RA18]), we first determine appropriate parameters in terms of tag and bucket size to reach negligible false positive probabilities of about 2^{-30} and about 2^{-40}. Then, we introduce a compression technique that skips empty buckets in transit and storage. The compression ratio therefore directly corresponds to the load factor of the Cuckoo filter. This is especially useful for fast-growing messengers as sparsely loaded filters can be distributed initially and then updated, instead of transferring a completely new filter once the maximum capacity is reached. The same compression technique was also proposed in concurrent

and independent work as part of so-called Morton filters [BJ18]. Furthermore, we propose an efficient update procedure that reduces communication by a factor of 4.3×: instead of transferring an encrypted element to the client to be inserted, we only transfer the tag and one of the possible insertion positions.

More Efficient PRF. Our GC-PSI protocol is based on the oblivious evaluation of a Boolean circuit representing a symmetric block cipher. In previous works [PSSW09; KLS+17], AES with a circuit size of 5,120 AND gates was chosen for this purpose. We suggest to replace AES with LowMC [ARS+15], a symmetric block cipher specifically optimized for oblivious evaluation with Multi-Party Computation (MPC) protocols. We determine suitable parameters (in terms of block size, key size, number of S-boxes, number of rounds, and data complexity) for LowMC such that at the same 128 bit security level, we require only 624 AND gates, which directly translates to a performance improvement of factor 8.2×. Our parameter choices are not affected by recent attacks [LIM21a; LIM21b] that focus on specific low-round configurations of LowMC as used in the post-quantum signature scheme Picnic [KZ20].

Malicious Security. The protocols by [KLS+17] that we build upon are only secure against semi-honest adversaries. However, we observe that the only messages sent by the client occur during OT. Therefore, we can secure our protocols against malicious clients by instantiating the OT part with maliciously secure protocols. For our implementation, we choose the base OT protocol of [CO15; DKLS18] and the OT extension protocol of [KOS15], which does not noticeably affect performance compared to semi-honest alternatives [ALSZ13]. Note that even for semi-honest clients it is nevertheless possible to manipulate the inputs for the PSI protocol. This issue of enumeration attacks was extensively discussed in §2.1.1.

Native Implementations. Implementations of OT and MPC protocols nowadays frequently rely on Intel's AES-NI instructions to accelerate seed expansion and garbling via fixed-key AES [BHKR13]. However, protocols for mobile private contact discovery are supposed to be executed on smartphones that are usually equipped with ARM SoCs instead of Intel CPUs. In our implementation, we therefore utilize the Cryptography Extensions (CE) that bring similar capabilities for all chips from the widely available ARMv8 architecture onwards. Comparing the garbled circuit implementation of [LWN+15; KLS+17] in Java with our native C/C++ implementation using ARM CE, we measure a run-time improvement of factor 1,000×. The ARM NEON instruction set furthermore provides vector operations, which behave similar to Intel's SSE and AVX instructions and which we can utilize for efficiently handling wire labels of garbled circuits. Additionally, we port x86 specific parts of `libOTe`[12] to the ARM instruction set.

Evaluation. We evaluate our protocols' implementation on a Google Pixel 2 XL smartphone with a Snapdragon 835 CPU clocked at 2.45 GHz and with 4 GiB of RAM that represents the client. A PC with an Intel i7-4600U CPU at 2.60 GHz and 16 GiB of RAM represents the server. Regarding the network connection, we study two scenarios: An IEEE 802.11ac Wi-Fi

[12] https://github.com/osu-crypto/libOTe

connection with 230 Mbit/s down- and upload bandwidth and 70 ms Round Trip Time (RTT), and a real LTE connection with 42 Mbit/s download, 4 Mbit/s upload, and 80 ms RTT. As a comparison, we run the protocols of [KLS+17] in our environment, which are the only other available implementation of unbalanced PSI protocols on actual mobile devices.

We depict the resulting run-times and communication overheads for both network settings for the setup phase with 2^{28} server contacts in Figure 2.2. The 1.3× improvement stems directly from our Cuckoo filter compression as we distribute a filter filled with 70 % of its capacity.

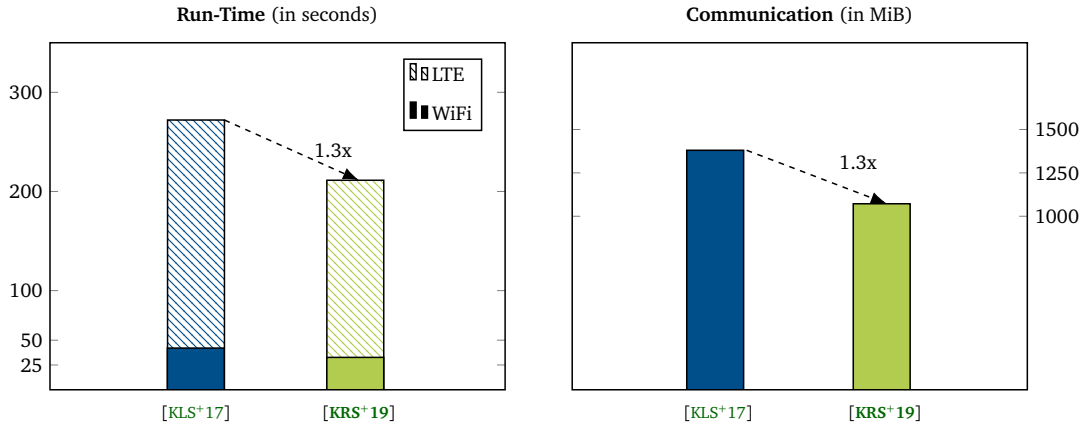


Figure 2.2: Comparison of setup phase for 2^{28} entries in the server database.

Likewise, we depict the benchmarks for the combined base and online phase when checking 1 k address book entries against the server database in Figure 2.3. Both of our protocols achieve significant improvements over prior work [KLS+17]. Compared to each other, both of our protocols deliver rather similar performance, with a slight benefit for the NR-PSI protocol in terms of communication overhead.

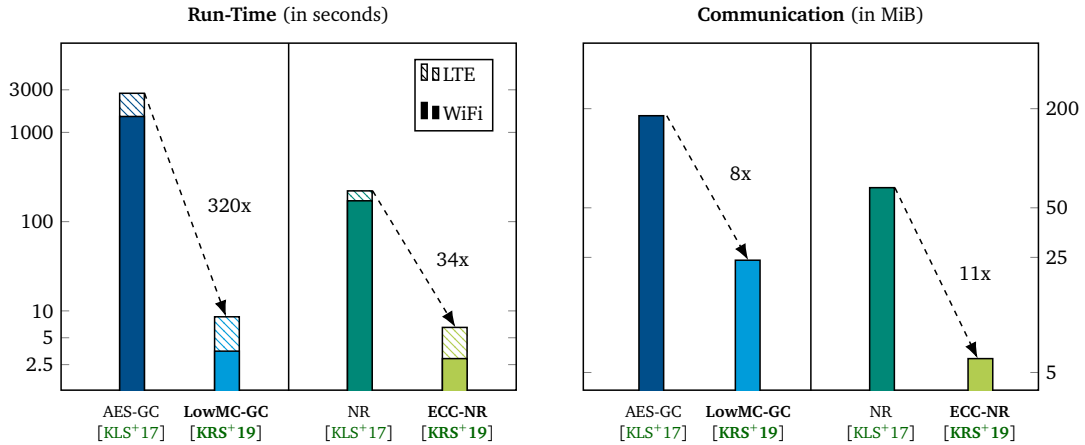


Figure 2.3: Comparison of combined base and online phase for checking 1,000 client contacts.

CogniCrypt Integration. One of the most common reasons for insecure software is the misuse of cryptography APIs that are hard to use securely for non-expert software developers [EBFK13; LCWZ14; CNKX15]. As a solution to this long-standing problem, we helped to develop the open-source platform CogniCrypt [**KNR+17**] as part of the Collaborative Research Center (CRC) Cryptography-Based Security Solutions (CROSSING) funded by the German Research Foundation (DFG). CogniCrypt is an Eclipse plugin[13] that for various standard cryptographic tasks (e.g., encryption of a file) offers a code generation wizard. In later steps, CogniCrypt also performs static code analysis to make sure the automatically generated template code is not modified in ways that make it insecure.

We added unbalanced PSI as an additional use case in CogniCrypt. The configuration wizard asks a series of simple questions (e.g., how many elements are expected on the server side to set Cuckoo filter parameters accordingly) and then automatically generates Java code for client and server side. This template contains code for the individual protocol phases and additional helper methods to, for example, manage the Cuckoo filter and store the server key. In terms of static analysis, we make sure that all protocol phases are executed in the intended order. Currently, our integration comes with support for Linux and Windows servers where we include our native code via a Java Native Interface (JNI) bridge. Beyond private contact discovery, developers with our CogniCrypt integration can also securely utilize our code for other applications of unbalanced PSI, e.g., privacy-preserving malware checking [KLS+17].

Combination with PIR / Non-Colluding Servers. When we contacted Signal to ask about chances for deployment of our PSI protocols, they returned an incredibly demanding requirement list in terms of run-time and communication overhead. Although being nowhere as popular as, e.g., WhatsApp, they do not assess solutions based on their current user base (which we estimate to be approximately 2.5 M in the US, cf. §2.1.1) but any solution must be able to efficiently handle up to one billion registered and active users. While our run-times as shown in Figures 2.2 and 2.3 already have the right proportions, the setup communication in the order of gigabytes for large-scale databases is far from Signal's wish of spending at most 10 MB for setting up contact discovery. Since we cannot meet these requirements, in [**KRS+19**] we present a combination with Private Information Retrieval (PIR) that allows to significantly reduce the communication overhead. In general, PIR protocols allow clients to obliviously query a server database such that the server does not learn which element was requested. In our construction, we suggest to transfer the Cuckoo filter to a second (non-colluding) server instead of the client. After obtaining encryptions of the address book entries via regular OPRF evaluations with the server of the service provider, the client then can run a multi-server PIR protocol with both servers to obliviously check whether the element is contained in the Cuckoo filter. When using a multi-server PIR protocol such as [BGI16], these checks have logarithmic instead of linear communication complexity in the size of the server database. However, the critical additional assumption is that the operators of both servers do not collude in the sense that they exchange the messages received as client queries. Collusion would allow the service provider to deduce which phone numbers the

[13] https://www.eclipse.org/cognicrypt

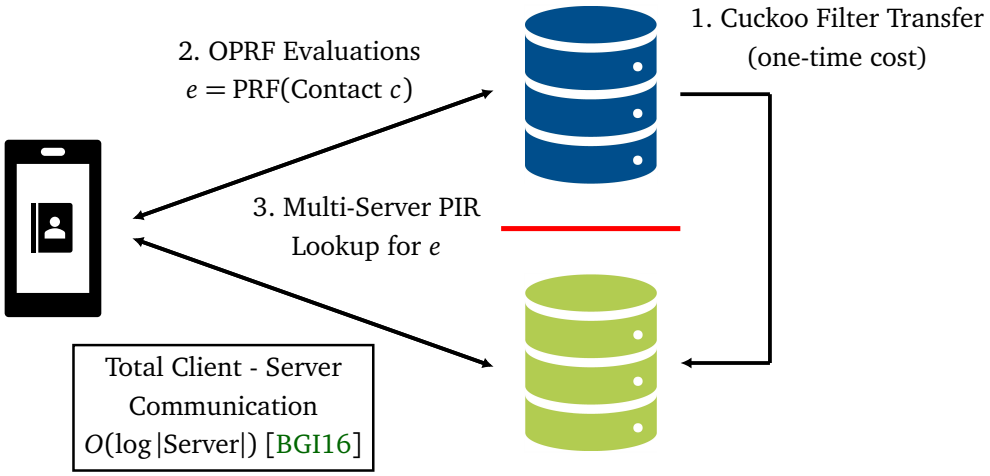


Figure 2.4: Combination of PSI protocols with multi-server PIR as suggested in [KRS+19].

client has checked, as is the case for currently deployed insecure contact discovery methods. Our construction is visualized in Figure 2.4.

Impact. We try to raise awareness in the general public for the privacy risks incurred by currently deployed insecure contact discovery methods. For this, we produced "explainer videos" in both English (https://youtu.be/4vgKHmNaAAw) and German (https://youtu.be/P_K166jvG7U) that educate users about the potential threats and on a high level describe the three step approach of our improved protocols as visualized in Figure 2.1. Furthermore, we maintain a project website at https://contact-discovery.github.io that is regularly updated with recent developments. Our work in [KRS+19] received the second prize in the 8. German IT-Security Award 2020 by the Horst Hörtz foundation[14], which is one of the most highly endowed awards in IT security for outstanding innovations of great practical and market relevance.

2.2 Related Work

In the following, we put our work on attacks on mobile contact discovery into perspective of prior research and recent events. Also, we more closely review previous works in terms of PSI protocols for mobile private contact discovery as well as alternative approaches from the area of secure computation.

[14] https://www.deutscher-it-sicherheitspreis.de

Study			[MZM+18]	[HWS+21]		
Method			Brute Force (hybrid)	Database Look-Ups	Brute Force (hybrid)	Rainbow Tables
Software			hashcat	Redis	hashcat	RainbowPhones
Hash Algorithms			MD5 / SHA-256	SHA-1		
Hardware			NVIDIA GTX 1050 Ti	630 GB of RAM	2× NVIDIA Tesla P100	Intel Core i7-9800X
Run-Time in (hh:mm:ss)						
Search Space	China	$(2.3 \cdot 10^{10})$	00:07:17 / 00:07:12	—	—	—
	Germany	$(4 \cdot 10^{11})$	02:28:24 / 02:34:16	—	—	—
	Indonesia	$(5.8 \cdot 10^{11})$	02:45:57 / 02:52:42	—	—	—
	Mobile World-Wide	$(1.18 \cdot 10^{11})$	—	00:01:40*	15:55:00	14:26:40*

Table 2.1: Comparison of reversal methods for batches of 1 million phone number hashes. The coarse-grained estimations for the size of the phone number spaces for China, Germany, and Indonesia stem from [MZM+18]. Run-times marked with * are scaled from measurements for batches of 10 k hashes as reported in [**HWS+21**].

2.2.1 Attacks on Mobile Contact Discovery

There exist several works that study hash reversal of Personally Identifiable Information (PII) such as phone numbers as well as crawling attacks on mobile messengers. Additionally, we discuss the possibility and the issue of tracking (stalking) users after discovery.

Hash Reversal Attacks. Hashing is commonly used to securely store passwords at rest. This has sparked interest in hash reversal techniques to "crack" captured hashes [Mar08], and popular open-source tools exist for the purpose (e.g., hashcat[15], John the Ripper[16], and RainbowCrack[17]). While the input domain and therefore the search space for passwords in theory can be arbitrarily complicated, a common issue in practice is that users tend to choose simple passwords that can be found in dictionaries or short number sequences that make cracking such password hashes feasible [YBAG04; Tat15]. For the same reason, hashing of PII such as phone numbers, email addresses, or MAC addresses is insecure [Mar14; DKCL18].

This also follows intuitively when determining an upper bound on the search space and then considering hash rates for current CPUs, GPUs, FPGAs, or specialized ASICs as heavily utilized to accelerate hashing for blockchain mining [OLe18]. According to the E.164 standard[18], phone numbers consist of at most 15 digits such that there are at most $\approx 1.2 \cdot 10^{15}$ possible phone numbers (ignoring that only specific country codes and area/provider prefixes are in use). On a system with two NVIDIA Tesla P100 GPUs, we achieve a SHA-1 hash rate of 9.5 GHashes/s according to benchmarks with hashcat. A brute-force attack should therefore be successful in at most 1.46 days. When considering our accurate database of mobile number prefixes (cf. §2.1.1), a brute-force attack should take no more than 12.42 s.

To not rest with rough estimations, there have been attempts to empirically quantify the effort for reversing phone number hashes [MZM+18]. We compare our findings with [MZM+18]

[15] https://hashcat.net/hashcat
[16] https://www.openwall.com/john
[17] https://github.com/inAudible-NG/RainbowCrack-NG
[18] https://www.itu.int/rec/T-REC-E.164

in Table 2.1. Our measurements confirm that simply applying standard tools for cracking phone number hashes does not deliver the expected performance, and estimations based on hash rate benchmarks are wrong by orders of magnitude. Especially when using hashcat's hybrid mode that brute-forces phone number hashes based on prefix masks, the hash rate drops significantly when choosing fine-grained masks to limit the search space due to the overhead of distributing workloads on the GPU. This can be seen in Table 2.1, where the coarsely-specified search space for German numbers ($4 \cdot 10^{11}$) of [MZM+18] is 6.45× faster covered on older hardware (NVIDIA GeForce GTX 1050 Ti) than our fine-grained prefix database for world-wide phone numbers ($1.18 \cdot 10^{11}$, cf. §2.1.1) on two NVIDIA Tesla P100 GPUs. Nevertheless, both works agree that phone number hash reversal is feasible on consumer hardware, and with look-ups in in-memory databases and our optimized rainbow table construction we also study two other methods that allow to reverse single phone number hashes on the fly.

Crawling Attacks. Over the years, there have been several attempts from researchers and white hat hackers to crawl various social networks and mobile messengers with enumeration attacks [BSBK09; BPH+10; SFK+12; CYJ+13; MSF+14; KPKS15; Gup16; GGAK16; KKC+17], with several (outdated) open-source crawling tools being available, e.g., for WhatsApp[19,20]. Many of these projects followed responsible disclosure, making service providers aware of the issues. Also, media reports resulted in public pressure to at least establish reasonable rate limits [Cox17; Dof19]. Therefore, we initially expected our experiments to fail in the sense that they produce no surprising results. Unfortunately (or luckily, depending on the perspective), this was not the case.

In Table 2.2, we summarize and compare earlier crawling attempts on popular mobile messengers (in addition to WhatsApp, Signal, and Telegram also KakaoTalk[21] and WeChat[22]) to our results discussed in §2.1.1. In comparison, we conduct crawling attacks at a much larger scale and provide concrete estimations for currently deployed rate limits.

When we responsibly disclosed our findings regarding WhatsApp in September 2019, our security report was initially dismissed: Facebook considered their existing protection measures sufficient to prevent data scraping and they see legitimate contact discovery requests from enterprises with more than 200 k employees (even though WhatsApp's terms of service forbid commercial use for regular accounts). Only later in October 2019, our report was re-opened upon personal interaction with security engineers, and fixes were finally deployed in July 2020. As it turned out, an extremely similar vulnerability in Facebook's core application has lead to a leak of 533 million users' data [Hol21], of which Facebook was aware even prior to our report. More precisely, Facebook's "contact importer", which suggests friends based on users' address book entries, was vulnerable to enumeration attacks and severely abused to perform large-scale crawling attacks as we described for WhatsApp [New21]. The exact chronology of events including our involvement is documented in a Business Insider article [Pri21].

[19] https://gitlab.com/jishnutp/whatsapp-crawler
[20] https://github.com/JMGama/WhatsApp-Scraping
[21] https://www.kakaocorp.com/service/KakaoTalk
[22] http://www.wechat.com/en

Study	Messenger	Method	# Numbers	Rate Limits
[CYJ+13]	WeChat	ADB + API Monitoring	100 k	—
[KPKS14; KPKS15]	KakaoTalk	Export Feature	101 k	—
		Optical Character Recognition (OCR)		
		Debugging Tool		
[SFK+12]	WhatsApp	—	10 M	None
[Klo17]		Chrome Extension	—	—
[**HWS+21**]		UI Automator	46.2 M	60 k / d
	Signal	(Legacy) API	505.7 M	120 k / d
	Telegram	API	0.1 M	5,000 + 100 / d

Table 2.2: Comparison of enumeration attacks on popular mobile messengers. Rate limits based on the experiences reported by the authors when querying the stated amount of phone numbers. "—" indicates no or unclear information provided by the authors.

User Tracking. Once the registration state of phone numbers with certain mobile messengers is confirmed via enumeration attacks, it is then possible to closely monitor users and track user behavior over time. While we do not investigate the feasibility and implications of such lateral attacks, previous works have developed tools such as "Online Status Monitor"[23] and "WhatsSpy" [Zwe15a; Zwe15b; Zwe16] for this purpose. The accuracy and power of such surveillance tools is well documented [BKN+14], and users rightfully express concerns about privacy-invasive features of mobile messengers such as the "Last Seen" timestamp that is displayed to indicate when a user has last opened the messenger app [CO13; RVC16]. A recent debate evolves around the fact that some messengers like WhatsApp do not even allow users to disable displaying their online status by editing the privacy settings, which so-called "stalkerware" exploits to help abusers closely monitor their victims [Sto21]. WhatsApp in response only suspends accounts used by tracking apps to monitor other users but does not plan to change the situation since they see always displaying the online status as intended behavior [Fra21].

2.2.2 PSI Protocols for Mobile Private Contact Discovery

In the following, we compare our PSI protocols for mobile private contact discovery with prior works in the area of unbalanced PSI. Additionally, we explore combinations of our and other PSI protocols with PIR to enhance performance under the assumption of non-colluding servers. Finally, we discuss Signal's TEE-based approach to provide mobile private contact discovery via Intel SGX.

Unbalanced PSI Protocols. Except for constructions based on computationally expensive Homomorphic Encryption (HE), cryptographic PSI protocols inherently require communication that is linear in the size of both input sets. In scenarios with unequal set sizes, where the party with the smaller input set is a mobile device, such protocols thus become

[23] https://onlinestatusmonitor.com

<table>
<tr><th rowspan="2">Protocol</th><th rowspan="2">Data Structure</th><th colspan="2">Communication</th><th rowspan="2">OPRF / Primitive</th><th colspan="2">Adversary Model</th></tr>
<tr><th>Setup</th><th>Online</th><th>Client</th><th>Server</th></tr>
<tr><td rowspan="3">[KLS+17]</td><td rowspan="3">Bloom Filter</td><td rowspan="4">$O(n)$</td><td rowspan="4">$O(m)$</td><td>RSA</td><td colspan="2" rowspan="4">Semi-Honest</td></tr>
<tr><td>NR-PRF</td></tr>
<tr><td>AES-GC</td></tr>
<tr><td rowspan="2">DH</td></tr>
<tr><td>[RA18]</td><td>Cuckoo Filter</td></tr>
<tr><td>[CLR17]</td><td rowspan="2">Hash Tables</td><td colspan="2" rowspan="2">$O(m \log n)$</td><td rowspan="2">Leveled FHE</td></tr>
<tr><td>[CHLR18]</td><td colspan="2">Malicious</td></tr>
<tr><td>[KRS+19]</td><td>Cuckoo Filter</td><td>$O(n)$</td><td>$O(m)$</td><td>NR-PRF
LowMC-GC</td><td>Malicious</td><td>Semi-Honest</td></tr>
</table>

Table 2.3: Comparison of unbalanced PSI protocols. n is the size of the (large) server set, m of the (small) client set.

impractical. Therefore, there have been attempts to at least get rid of transferring the larger set during the online phase of PSI. Instead, most of the computation and communication overhead is shifted to a precomputation phase that can be carried out at an arbitrary point in time before the actual PSI query. Especially the work of [KLS+17] has pioneered the notion of "unbalanced" PSI by converting four PSI protocols into this precomputation form based on RSA blind signatures [CT10], Diffie-Hellman key exchange [HFH99], the Naor-Reingold-PRF [NR04; HL10], and a garbled circuit evaluation of AES [PSSW09]. These protocols follow the idea and flow visualized in Figure 2.1. In our work [**KRS+19**], we improve two protocols that can be trivially adapted to provide security against malicious clients: the one based on the Naor-Reingold-PRF (NR-PSI) [NR04; HL10] and the garbled circuit evaluation of AES (AES-GC) [PSSW09].

Following [KLS+17], the work of [RA18] brings the DH-style protocol of [BBC+11] in precomputation form and introduces Cuckoo filters [FAKM14] as a probabilistic data structure to represent and distribute the encrypted server set. In [**KRS+19**], we adapt the idea of using Cuckoo filters, but set realistic parameters in terms of false positive probability, introduce Cuckoo filter compression (as also proposed in concurrent and independent work [BJ18]), and provide more efficient Cuckoo filter updates.

Finally, the works of [CLR17; CHLR18] have studied PSI for unbalanced set sizes based on (leveled) Fully Homomorphic Encryption (FHE). In these protocols, the client first encrypts the items of the small input set under its own public key, the server (under encryption) subtracts the items of its large input set from each received item, and returns the product of the results randomized such that if decryption results in 0, a match was found; otherwise, no information is leaked. For efficiency improvements, the protocols follow the hashing paradigm of [PSZ18] for bin-wise operations. While [CLR17] is restricted to a fixed bit length of 32 bit, [CHLR18] adds support for arbitrary bit lengths and provides security against malicious adversaries. The benefit of the FHE-based approach compared to other unbalanced PSI protocols is that the overall communication is sub-linear in the size of the larger server set.

However, the computation cost for processing each item in the server set under encryption for each PSI session is too demanding to be deployed by messaging services: in [**KRS+19**] we estimate that a service provider would need to pay for 28.9 M core hours per day to operate a contact discovery service with a database size of 2^{28} and having the same amount of active users perform one synchronization per day. In contrast, the major cost factor when deploying protocols as described in [**KRS+19**] at a similar scale stems from initially handling 268 PiB of network traffic for sending a Cuckoo filter with a size of approximately 1 GiB to each registered user during the setup phase. Potentially, this cost can be significantly alleviated by utilizing peer-to-peer distribution similar as done for updates in Microsoft Windows [OCS+21].

In Table 2.3, we summarize this comparison of unbalanced PSI protocols.

Combination of PSI with PIR. A similar combination of PSI with Private Information Retrieval (PIR) as we described in §2.1.2 was proposed under the name "PIR-PSI" in prior work [DRRT18]. The standard variant of [DRRT18] considers two non-colluding servers where both servers know the user database in the clear. In contrast, in our proposal, the second server only holds a Cuckoo filter containing the encrypted database. In PIR-PSI, the client then uses a modified multi-server PIR protocol [BGI16] to query the positions for their input set in a hash table on the server side. Due to the modification, one of the servers learns the result of the PIR query but masked with values known to the client. The client and the second server can then run the PSI protocol of [KKRT16] on the masked values to determine the actual intersection. The evaluation conducted by [DRRT18] states a total communication of $\approx$ 32 MiB to query more than 10 k contacts in a server database with 2^{24} entries. Due to the logarithmic growth of required communication, the overhead for larger databases with more than a billion entries would be moderate (although above the 10 MB requested by Signal). The exact communication overhead and concrete run-time performance of our combination of PSI with PIR has yet to be evaluated as part of future work.

As we see no hope to further optimize regular PSI protocols to the extent that they would satisfy Signal's requirements, combinations of PIR with PSI are a reasonable and viable alternative. However, to quote from a personal email exchange with Moxie Marlinspike, co-founder of Signal: "I believe that 'non-colluding servers' don't exist, will never exist, and imo should be completely removed from any discourse in the realm of cryptography." Therefore, chances are unfortunately low that Signal deploys a combination of multi-server PIR with PSI in order to enable mobile private contact discovery. On the contrary, the Internet Security Research Group (ISRG), parent organization of the popular non-profit certificate authority Let's Encrypt, recently announced [AG20] to operate a reliable service that acts as the non-colluding party required for privacy-preserving aggregation of user metrics with the "Prio" system [CB17]. Thus, we hope that at least other messaging services might consider the adaptation of our PIR-based solution in the future.

Trusted Execution Environments. Instead of a cryptographic protocol as discussed above, Signal in 2017 launched a technology preview for conducting private contact discovery using Intel's TEE implementation SGX as a potential replacement for the insecure exchange

of truncated phone number hashes [Mar17]. As SGX does not inherently protect against software side-channel attacks, developers have to mitigate such risks themselves. Signal's enclave implementation[24] therefore incorporates Oblivious Random Access Machine (ORAM)-style techniques for a branch-free and memory access pattern-hiding comparison of client contacts against the database of registered users. However, side-channel attacks have been presented that do not require vulnerabilities in the victim enclave to extract sensitive data (e.g., [BMW+18]). Therefore, while Signal's Intel SGX-based solution can be considered as a reasonable best effort, it is questionable whether users' privacy is significantly improved in the long term.

[24] https://github.com/signalapp/contactdiscoveryservice

3 Privacy-Preserving Authentication for Apple AirDrop

AirDrop is one of Apple's proprietary wireless protocols. It enables convenient (offline) file sharing between nearby Apple devices such as iPhones and MacBooks running iOS and macOS, respectively. Introduced in 2011, this feature is currently available on more than 1 billion active devices [Kas21]. The flow of sharing a file using AirDrop is as follows: The sender opens the sharing pane on the sending device. In the sharing pane, a selection of discovered potential receivers is presented. After selecting the preferred receiver, the receiver is asked to accept the incoming request. If approved, the file is transferred.

By default, as users typically want to receive requests only from people they know, AirDrop devices can be discovered by "contacts only": in this mode, devices only respond to discovery requests if the receiver knows the sender, i.e., has a phone number or email address of the sender stored in the address book. Users can furthermore change the settings to be discovered by "everyone" or completely turn "receiving off". To accelerate the sender's decision which potential receiver to select, AirDrop presents discovered devices in the sharing pane including contact name and picture – if the sender has one of the receiver's phone numbers or email addresses stored in the address book. Otherwise, only the device name is displayed. Implementing the "contacts only" mode for receivers and mapping nearby devices to known contacts for the sender requires some kind of authentication mechanism to establish whether sender and receiver are mutual contacts.

Under the hood, AirDrop is based on a combination of Bluetooth Low Energy (BLE) and the Apple Wireless Direct Link (AWDL) protocol [SKH18a] to find suitable receivers and to transfer files. To facilitate analyses of Apple's proprietary protocols, prior work [SNM+19] has reverse-engineered AirDrop as part of the OpenWireless project [SKH18b]. The results are available also in form of the open-source Python implementation OpenDrop [HS20]. In the following, we summarize the protocol flow of AirDrop based on [SNM+19] with a focus on the authentication procedure.

Upon opening the sharing pane on an AirDrop-enabled device, Bluetooth advertisements containing the first two bytes of the SHA-256 hashes of the sender's contact identifiers (which are phone numbers and email addresses registered with the Apple ID account and thus verified by Apple) are broadcasted. If in "everyone" mode or if one of the hash prefixes matches a contact book entry of the receiver, the AWDL interface is activated and a TLS connection is established to run the authentication protocol. In this protocol, the sender in the initial HTTPS "discover" message transfers a so-called validation record, which contains the full 256 bit output of SHA-256 hashes of the sender's contact identifiers and is signed by Apple. If the validation record has a valid signature and the receiver's address book

contains a contact with an identifier that matches one of the hash values included in the record (i.e., the receiver knows the sender), the receiver proceeds with the authentication by responding with their own validation record. Otherwise, the authentication phase is aborted and the connection treated as unauthenticated. The sender, upon receiving a validation record with a valid signature, can likewise determine whether they know the receiver by checking the embedded hash values against their own address book entries. If all checks succeed, the connection is authenticated and the receiver is shown in the sender's sharing pane with their contact name and picture.

The work of [SNM+19] has shown that the use of truncated hashes as an initial authentication measure in the Bluetooth advertisement phase is insufficient: malicious senders can quickly trick devices in "contacts only" mode to activate their AWDL interface via brute force. In the following, we study privacy issues regarding the use of hashed contact identifiers in the subsequent authentication protocol and propose suitable mitigations.

3.1 Our Contributions

This thesis has contributed significantly to uncover and mitigate privacy vulnerabilities in the authentication phase of Apple's AirDrop protocol with the following publications that can be found in Appendices C and D:

[HHS+21a] A. Heinrich, M. Hollick, T. Schneider, M. Stute, C. Weinert. **"DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop"**. In: *14. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/893. ACM, 2021, pp. 371–373. Appendix C.

[HHS+21b] A. Heinrich, M. Hollick, T. Schneider, M. Stute, C. Weinert. **"PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop"**. In: *30. USENIX Security Symposium (USENIX Security'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/481. USENIX Association, 2021, pp. 3577–3594. CORE Rank A*. Appendix D.

Specifically, we find that the exchange of vulnerable hash values during the authentication handshake leads to leaking the contact identifiers of the AirDrop sender and in some cases also the receiver, which we practically show with our demonstrator "AirCollect" [**HHS+21a**]. Based on these findings, we develop our privacy-preserving mutual authentication solution "PrivateDrop" [**HHS+21b**], for which we integrate an optimized version of a maliciously secure PSI protocol [JL10] and show with a native prototype implementation on iOS and macOS devices that we can achieve practical performance with an overall authentication delay well below one second to preserve AirDrop's user experience.

Privacy Vulnerabilities. In the mutual authentication phase of Apple's AirDrop protocol as described in §3, sender and receiver exchange Apple-signed validation records that contain SHA-256 hashes of their Apple-verified contact identifiers (phone numbers and email addresses). As we have shown in [**HWS+21**] in the context of mobile contact discovery (cf. §2.1.1), the entropy of mobile phone numbers is extremely low such that almost instant hash reversal is possible even on consumer-grade hardware. Likewise, for reversing hashes of email addresses, it is possible to utilize commercial web services that offer to reverse email addresses with 70 % success rate at the cost of 0.04 USD per address [Dat19]. This leads to two vulnerabilities that we refer to as "sender leakage" and "receiver leakage".

Sender leakage comes from transferring the user's validation record including hashed contact identifiers to all potential receivers unconditionally whenever the user opens the sharing pane on an iOS or macOS device. Therefore, adversaries with a Wi-Fi-enabled device in proximity can obtain the sender's hashed contact identifiers and apply hash reversal techniques to learn the contact identifiers in the clear. Especially "VIPs" must thus be careful to not open the sharing pane in public as they otherwise leak at least their private mobile phone number. Also, it is possible to plant small "bugs" in public hot spots to create a large-scale database of mappings from contact identifiers to specific locations. Such data can in later steps be abused, e.g., for targeted spear-phishing attacks.

Receiver leakage refers to the receiver's response in the authentication protocol containing their validation record including hashed contact identifiers – if they know the sender. However, it is not a requirement for the sender to know the receiver. This vulnerability can be exploited in two different ways: A person whose phone number or email address is stored by many people (e.g., a supervisor in a large company) can therefore learn the personal contact details of other employees while walking around. Unless the victims have changed their AirDrop setting to "receiving off", there is nothing they can do to defend against this attack as there is no user interaction required on the receiver side. For the second exploit variant it is important to note that the validation record contains hashes for *all* contact identifiers. Therefore, a sender who knows nothing or only partial information about the receiver can learn all remaining contact identifiers. This can be exploited, e.g., by a journalist who interviews a celebrity that has the journalist's email address stored as a contact. In this scenario, the journalist can additionally learn the celebrity's private mobile phone number.

Demonstrator "AirCollect". To practically show the severity of the discovered attacks, we implemented them in the demonstrator "AirCollect" [**HHS+21a**] to exploit sender as well as receiver leakage. Our open-source implementation is based on OpenDrop [HS20] to listen to and to trigger AirDrop traffic, and RainbowPhones [**HWS+21**] (cf. §2.1.1) to immediately reverse received hash values of mobile phone numbers with an optimized rainbow table implementation. AirCollect can turn any MacBook into a bug for passively collecting contact identifiers of senders or to actively exploit receiver leakage. For this, we extended OpenDrop with a command line interface to conveniently launch the attacks and the capability to parse validation records. The extracted hash values are then passed to RainbowPhones for cracking, which we extended with support for SHA-256 hashes and precomputed rainbow tables for

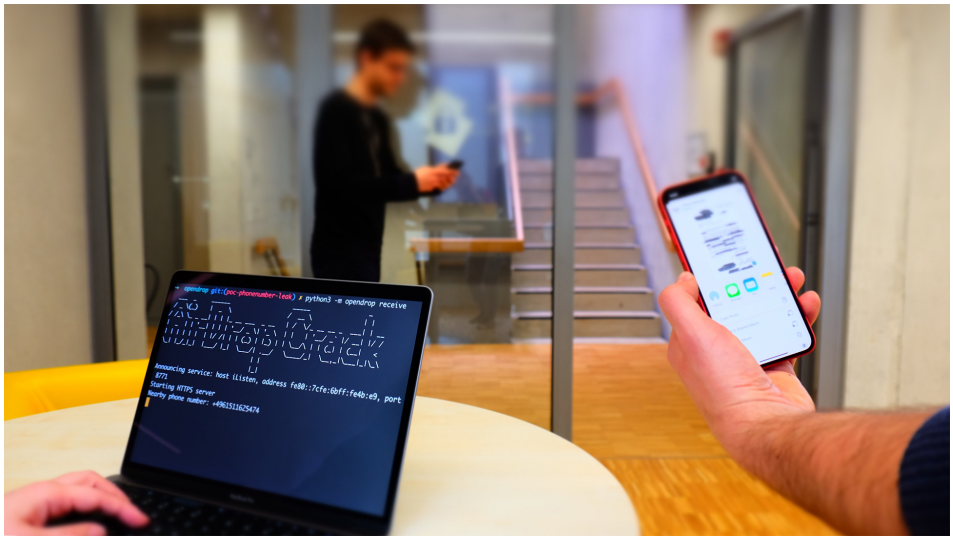


Figure 3.1: Test setup for our demonstrator "AirCollect" [HHS+21a]. The sender (right) intends to share a document with the receiver (background). The MacBook running AirCollect (left) immediately displays the sender's cleartext phone number.

all mobile phone numbers in Germany (76.5 MB with 99.8 % success rate) and all phone numbers in the US (765 MB with 100 % success rate). A picture of our test setup is shown in Figure 3.1.

Applicability of PSI. We observe that sender and receiver leakage in the authentication phase of AirDrop can be prevented by replacing the exchange of vulnerable hash values with two consecutive executions of a cryptographic PSI protocol. In the first PSI execution, the sender can input their contact identifiers and the receiver their address book such that the receiver learns "I know the sender" (in case of a non-empty intersection) or aborts. Then, we can run PSI with the roles reversed, such that the sender can say "I know the receiver" or aborts. This application of the PSI functionality is depicted in Figure 1.4 in §1.1.

However, we notice there are two issues of this straight-forward PSI application: The first is that a malicious receiver not necessarily aborts if the intersection in the first step was empty and can try to fool the sender by using widely popular phone numbers as input instead of the own ones, for example, emergency numbers that the sender has stored with high probability. The second problem is that the computation complexity of the online phase of the PSI protocol in this scenario mainly depends on the size of the address book, which is usually much larger than the set of contact identifiers registered by a user.

Based on a systematic evaluation of all possible input and role combinations, we therefore propose to slightly change the semantics by exactly swapping the inputs provided by each of the parties. This way, in the first execution, receivers only learn whether they are known by the sender, and then have to prove that they know the sender. This change also has the

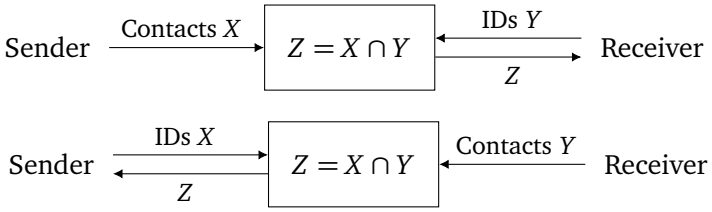


Figure 3.2: Ideal functionality for PSI when applied to mutual authentication for Apple AirDrop with our modified semantics. IDs is a set of the user's own phone numbers and email addresses, whereas contacts is a set of phone numbers and email addresses stored in the user's address book.

benefit that the online computation complexity of PSI now mainly depends on the smaller input set consisting of registered contact identifiers, which we assume to be at most of size 20. Once both protocol executions are done, the two parties can safely disclose their identifiers in later steps of the AirDrop protocol as they are known by the respective other party anyway. Our application of PSI to the AirDrop authentication phase is shown in Figure 3.2.

Optimized PSI Protocol. To realize the ideal PSI functionality utilized in the abstract protocol design shown in Figure 3.2, we have to choose a suitable protocol from the literature. The available options are extensively reviewed in §2.2.2 and §3.2.3. We choose the maliciously secure public-key-based PSI protocol of [JL10] that for comparatively small set sizes can be efficiently implemented using elliptic curve cryptography.

For the use in AirDrop, we modify the protocol to precompute operations that incur a complexity that is linear in the size of the larger input set (i.e., the address book) ahead of time to obtain an efficient online phase. Furthermore, we suggest to reuse these precomputed values across sessions to prevent frequent initialization efforts and to bundle/interleave the messages for the consecutive executions to reduce the number of communication rounds. Finally, in order to prevent that users lie about their contact identifiers to cause fake matches, we additionally propose to use signed inputs in the protocol as proposed in [CZ09; CKT10; CT10]. For this, we suggest to leverage Apple's existing Certification Authority (CA) infrastructure to certify the authenticity of encrypted contact identifiers. These signatures can then be transferred as part of the PSI protocol and be verified by the respective other party before proceeding with the protocol.

Implementation and Integration. As we cannot directly modify the original AirDrop implementation to integrate PSI, we first re-implement AirDrop in Apple's native programming language Swift. Then, we include an implementation of our optimized PSI protocol based on the standard elliptic curve P-256 provided by the Relic library [AGM+18] to create a PrivateDrop prototype that can be used for realistic performance benchmarks. Additionally, we describe the necessary steps for Apple to actually turn our PrivateDrop prototype into a production-ready implementation, including a discussion on how to maintain backwards-compatibility with devices running the insecure AirDrop version.

Evaluation. We empirically evaluate our PrivateDrop implementation and compare it to the insecure AirDrop protocol as a base line. For this, we test various input sizes in terms of number of address book entries as well as registered contact identifiers, and also measure the influence of the PSI operations on the overall protocol execution. When running PrivateDrop between a MacBook Pro 2019 and an iPhone 12 mini for 15 k address book entries and 10 contact identifiers on each side, the authentication delay is below 500 ms. As any response under 1 s is perceived as immediate by humans [CRM91], we can therefore successfully claim to provide a practical privacy-preserving alternative to the original AirDrop protocol. The online phase of our PSI protocol actually accounts only for 5 % percent of the overall authentication delay. We separately measure the necessary precomputation that can be conducted independently of the other party at an arbitrary point in time before the online phase of the authentication protocol, which takes only 4 s on the iPhone 12 mini for 15 k address book entries and can be reused across sessions.

Impact. We responsibly disclosed the discovered privacy vulnerabilities to Apple in May 2019 and proposed our solution "PrivateDrop" in October 2020. While Apple acknowledges our findings, until now there is no indication whether they plan to deploy mitigations for the reported AirDrop vulnerabilities or adapt our PrivateDrop solution. Therefore, more than 1 billion devices are still vulnerable to the outlined attacks, which can be conveniently conducted using our "AirCollect" demonstrator that is publicly available at `https://encrypto.de/code/aircollect`. The prototype implementation of PrivateDrop is available as open source at `https://encrypto.de/code/privatedrop`. We additionally maintain a project website at `https://privatedrop.github.io` to inform the general public in a comprehensible manner about recent developments concerning AirDrop's privacy and our research. Due to the large number of affected devices that are still vulnerable to our privacy attacks, our work received significant media coverage, which is documented at `https://encrypto.de/news/privatedrop`.

3.2 Related Work

In the following, we first discuss an independent research project that concurrently discovered one of the privacy vulnerabilities in AirDrop's authentication protocol and published corresponding exploit code [Cha19] (cf. §3.2.1). The closest related work to our secure alternative PrivateDrop proposed a mutual authentication protocol for Apple AirDrop based on Identity-based Encryption (IBE) [WTSB16]. In §3.2.2, we explain their protocol and detail how their study significantly differs from ours. Finally, in §3.2.3, we review PSI protocols from the literature that we considered as candidates when designing PrivateDrop.

3.2.1 Apple Bleee Project

In July 2019, two months after our initial disclosure with Apple, the penetration testing company Hexway published a blog post about their so-called "Apple Bleee" project [Cha19].

As part of this project, they describe multiple issues in the Apple wireless ecosystem that leak sensitive user data, for example in Apple's Wi-Fi password sharing feature. Furthermore, they describe a privacy vulnerability in Apple AirDrop that equals what we call sender leakage. For this, they also create a proof-of-concept implementation that is available on Github at `https://github.com/hexway/apple_bleee` that suggests to reverse phone number hashes based on lookup tables.

In contrast to the Apple Bleee project [Cha19], we verifiably followed the official responsible disclosure procedure before their publication, additionally discovered issues around so-called receiver leakage, implemented both attacks as part of "AirCollect" based on optimized rainbow tables for efficient phone number hash reversal [**HHS+21a**], and proposed a viable mitigation technique with our PrivateDrop protocol [**HHS+21b**].

3.2.2 IBE-based Private Mutual Authentication for Apple AirDrop

In [WTSB16], the authors observe a privacy leak in Apple AirDrop as deployed in iOS 9 that stems from the users' iCloud identity being included in the clear in the TLS certificate that is used to set up an authenticated connection between devices after the initial announcement of truncated hash values via BLE. Therefore, even a completely passive adversary can collect the identities of nearby Apple users if one of them opens the sharing pane and connects with potential receivers. As a mitigation, the authors of [WTSB16] design a zero round-trip private mutual authentication protocol (similar to [Aba02; AF04; JL09b]) based on Identity-based Encryption (IBE). An IBE scheme is an asymmetric cryptographic system where the public keys are derived from identities, e.g., strings that represent names [Sha84; BF01].

The private mutual authentication replacement for AirDrop proposed by [WTSB16] based on IBE works as follows: when a potential receiver finds a match for the advertised truncated hash values of the sender's contact identifiers, the receiver transfers an encryption of the own identity under an authorization policy that allows only that particular sender to decrypt. This requires Apple to act as the root authority in the IBE scheme and to provision secret keys for the contact identifiers of all registered users. The authors also implement their approach on an Android mobile phone (Google Nexus 5X) and report an overall authentication delay of 360.4 ms excluding networking.

First, we note that AirDrop at least from iOS 12 on is implemented differently as shown by [SNM+19]: the TLS certificates used for client authentication when setting up a secure communication channel do not include the user's iCloud identity but instead an account-specific Universally Unique Identifier (UUID). Such a UUID is still vulnerable to user tracking but cannot be directly mapped to an identity. Therefore, an entirely passive eavesdropper cannot learn the identities of nearby AirDrop senders and receivers.

Nevertheless, the scheme of [WTSB16] could be integrated into AirDrop to perform the mutual authentication step in a privacy-preserving way. However, compared to our solution PrivateDrop [**HHS+21b**], this approach has multiple disadvantages: To provision secret

keys for the IBE scheme and to establish corresponding certificate chains, Apple must introduce an IBE infrastructure, whereas we integrate PrivateDrop into Apple's existing CA infrastructure. Furthermore, we only require Apple to attest and sign locally generated encryptions of contact identifiers but not to centrally issue secret keys. Additionally, it is important to consider that contact identifiers such as phone numbers change from time to time. Thus, the IBE-based protocol must extended to support efficient revocation [BGK08], for which there exist established methods in regular Public Key Infrastructures (PKIs).

While the protocol of [WTSB16] protects against sender leakage, it is, to the best of our understanding, still possible for senders to learn the receiver's identity without knowing them in advance: when a receiver sends their identity encrypted under a policy such that only the owner of a secret key corresponding to the sender's contact identifier can decrypt, there is no requirement for the sender to have prior knowledge about the receiver's identity. In contrast, with our proposed modification to the semantics of the mutual authentication phase in AirDrop, the sender in the first PSI protocol execution has to convince the receiver to already know the receiver.

Regarding the evaluation, we provide a prototype implementation on real iOS and macOS devices and conduct our performance measurements over the network protocols. As we show in [**HHS+21b**], at most half of the total run-time is consumed by cryptographic operations, whereas the rest is caused by networking delays. Since [WTSB16] exclude networking entirely from their measurements, it is unclear how their protocol performs when being run over AWDL. Also, as [WTSB16] uses truncated hashes from the BLE advertisements to select a potential sender's contact identifiers, multiple matches might be found in the receiver's address book due to a non-negligible collision probability. Thus, the cryptographic operations of the authentication protocol must in the worst case be repeated multiple times and the overall authentication delay can easily exceed the 1 s barrier after which humans notice unpleasant delays [CRM91].

3.2.3 Public-Key-based PSI Protocols

After defining the protocol flow for mutual authentication in AirDrop based on PSI, we had to choose a suitable PSI protocol from the literature for the instantiation. There exist very efficient PSI protocols based on Oblivious Transfer (OT) [PSZ14; PSSZ15; KKRT16; PSZ18; PRTY19; PRTY20], protocols optimized for the use case of unbalanced input set sizes (as extensively discussed in §2.2.2) [CLR17; KLS+17; CHLR18; DRRT18; RA18; **KRS+19**], and simple public-key-based protocols that have been proposed since the 1980's [Sha80; Mea86; JL09a; CKT10; CT10; JL10; BBC+11; RA18].

However, protocols from the first two categories (OT-based and unbalanced PSI protocols) gain their performance mainly from shifting communication overhead to an input-independent setup phase. During this setup phase, interaction between the parties is required, which is why such protocols are not suitable for our use case where devices meet ad-hoc. Also, some of them have assumptions about non-colluding servers [DRRT18] or rely on advanced cryptographic

Protocol	Security Model	Cryptographic Assumptions	Precomputation Form [KLS+17]
[CT10]	Semi-Honest	One-More-Gap-DH + ROM	No
[CT10] (blind-RSA)		One-More-RSA + ROM	
[CKT10]	Malicious	DDH + ROM	
[JL09a]		q-DHI + CRS	
[JL10]*		One-More-Gap-DH + ROM	
[BBC+11]	Semi-Honest		
[RA18]			Yes

Table 3.1: Comparison of linear complexity public-key-based PSI protocols. For our work in [**HHS+21b**] we choose the protocol of [JL10] marked with *.

primitives [CLR17; CHLR18] for which only academic prototypes but no industry-grade implementations exist (to the best of our knowledge, not even for OT extension protocols there is a production-ready implementation available).

Since we want to keep the barriers for real-world deployment of our solution low, we therefore closer inspect simple public-key-based protocols [Sha80; Mea86; JL09a; CKT10; CT10; JL10; BBC+11; RA18], which can be implemented easily using standard production-ready cryptographic libraries. While such protocols are often considered impractical due to their high computational overhead caused by costly public-key operations such as modular exponentiations, we experimentally confirm that for comparatively small set sizes (e.g., the set of own contact identifiers can be reasonably assumed to be of at most size 20) they are sufficiently efficient [**HHS+21b**].

In [CT10], the authors propose two semi-honest PSI protocols with linear computation as well as communication complexity: one under the One-More-Gap-DH (OMGDH) assumption and a variant using blind-RSA under the One-More-RSA assumption [BNPS03]. However, since in our use case of mobile clients we have to accommodate potential malicious user behavior, the follow-up work of [CKT10] is more interesting, where the authors enable protection against malicious adversaries under the regular Decisional Diffie-Hellman (DDH) assumption in the Random Oracle Model (ROM). The first linear-complexity PSI protocol with security against malicious adversaries was previously proposed in [JL09a] under the q-Diffie-Hellman Inversion (q-DHI) assumption. However, this protocol requires a trusted third party to precompute a safe RSA modulus in the Common Reference String (CRS) model.

In our work, we select the maliciously secure protocol of [JL10] under the OMGDH assumption in the ROM. More precisely, the protocol is maliciously secure considering adaptive queries. However, the authors note in [JL10] that every efficient adversary must be committed to all its inputs, which is why the notion can be reasonably assumed to be equivalent to the regular intersection functionality. In comparison to the work of [CKT10], the protocol of [JL10] requires 25 % less exponentiations on the receiver side and only one instead of two zero-knowledge proofs. It can be implemented efficiently using elliptic curve cryptography libraries, for which there exist plenty of production-ready options. Especially interesting

in this context is Apple's own CryptoKit[1] library. However, since CryptoKit currently does not (yet) expose the necessary low-level arithmetic operations to the developer, we base our prototype implementation on the portable Relic library [AGM+18]. In case Apple decides to replace AirDrop with PrivateDrop, they could of course extend CryptoKit accordingly.

[BBC+11] created a simplified semi-honest version of [JL10] that omits the zero-knowledge proof required to prevent/detect malicious behavior. Later, [RA18] transformed the protocol of [BBC+11] into the precomputation form of [KLS+17] for unbalanced PSI by storing the encrypted inputs of the PSI sender in a Cuckoo filter [FAKM14] that is transferred to the PSI receiver ahead of time. In principle, it would be possible to use a probabilistic data structure like a Cuckoo filter in our PrivateDrop protocol to compress the encrypted input set of the PSI sender. However, since we operate in a setting where the maximum size of the encrypted input set is well below 1 MB even for address books with more than 10 k entries and we transfer data over a high-bandwidth direct Wi-Fi connection, the benefit of a smaller transmission size can be assumed to be negligible. At the same time, depending on the parameters, there is the chance to introduce false positives in the matching procedure when utilizing probabilistic data structures.

We summarize and compare the discussed public-key-based PSI protocols in Table 3.1. Note that here we do not consider other previously proposed related works with super-linear computation or communication complexity, for example, [FNP04; KS05; DMRY09; HN10].

[1] https://developer.apple.com/documentation/cryptokit

4 Privacy-Preserving Database Intersection Analytics

Due to increasing digitization, customers and citizens with every (trans)action leave traces in databases of companies and government agencies. For such organizations it could in many cases be beneficial to learn certain statistics about common data subjects and draw conclusions from that. For example, calculating whether customers of company A in total spend more money than a certain threshold on products from company B. If so, the companies could decide to intensify their business relationship and advertise the products that are most popular with common customers in a bundle. Conducting such analyses on joint data subjects is what we refer to as PSI analytics or database intersection analytics.

However, the naive approach of simply exchanging lists of data subjects puts the individuals' privacy at risk as they can therefore be accurately traced and profiled across organizations, and might even be prohibited by privacy regulations such as the European GDPR [Eur16]. Also, from the organizations' perspective their databases represent a valuable asset that they do not want to share with third parties entirely to conduct explorative analyses. Instead, they want to only learn certain aggregate statistics or functions on the data subjects that they actually have in common. In other words, they want to utilize data for a specific, well-defined purpose without disclosing unnecessary details, especially about data subjects not represented in the intersection. This is what we refer to as *privacy-preserving* database intersection analytics.

A natural way to realize database intersection analytics in a privacy-preserving manner is to apply *generic* or *circuit-based* PSI (cf. §1). In circuit-based PSI, the intersection between two sets is obliviously computed in a generic MPC protocol such as Yao's garbled circuits [Yao82; Yao86] or the GMW protocol [GMW87]. Since such MPC protocols can obliviously compute any function that can be represented as a Boolean circuit consisting of only AND and XOR gates, it is trivial to extend the plain intersection computation with an analytics function that should be computed on top. This way, it is also not necessary to release the intermediate intersection, which often can be sensitive information itself, but only the final aggregated analytics result.

Besides the need for variants of PSI in practice, circuit-based PSI in general has several advantages over specialized PSI protocols: There exist many frameworks that implement generic MPC protocols (e.g., ABY [DSZ15] and our Chameleon framework [**RWT+18**]), so developers can rely on an existing code base. Furthermore, new optimizations for generic MPC protocols automatically translate to performance improvements for circuit-based PSI. For example, very recently, the "slicing and dicing"-based garbling scheme of [RR21] unexpectedly improved over the state-of-the-art "half-gates" garbling of [ZRE15] in terms of required

communication per AND gate. Regarding adaptability to various analytics functions, software developers can further rely on existing compilers that are specifically designed to convert high-level code written in domain-specific or regular programming languages such as C/C++ into circuits that are optimized for MPC. Notable examples for such circuit compilers include Fairplay(MP) [MNPS04; BNP08], HyCC [BDK+18] with our optimization for arithmetic decomposition [**DKS+21**], which is especially useful when compiling analytics functions that include arithmetic calculations, as well as our LLVM-based compiler toolchain that supports multiple high-level programming languages [**HST+21**]. As these circuits are then obliviously evaluated with established MPC protocols, software developers can confidently rely on existing security proofs and thus rapidly experiment with new variants.

A naive way to construct the basic circuit that computes the intersection between two sets (excluding following analytic steps) is to compare every item in the first set with every item in the second set. However, such a circuit has a complexity of $O(n^2)$, where n is the size of the input sets. Since the number of gates that must be obliviously evaluated within MPC cause computation and communication overhead, it is critical to reduce this complexity to achieve practical privacy-preserving intersection analytics for large-scale databases. More precisely, while evaluating XOR gates is "free" in Yao's garbled circuits and the GMW protocol [KS08], every AND gate requires computation (e.g., 4 AES evaluations on the garbler's and 2 AES evaluations on the receiver's side in Yao's garbled circuits [ZRE15]) and communication (e.g., 2κ bits in Yao's garbled circuits [ZRE15], where κ is the symmetric security parameter). The very recent results of [RR21] even reduce the required communication in Yao's protocol to $1.5\kappa+5$ bits per AND gate at the cost of slightly increased computation (at most 6 and 3 AES evaluations on the garbler's and receiver's side, respectively).

An improved version of the naive approach with $O(n^2)$ complexity is known as Sort-Compare-Shuffle (SCS) [HEK12], which operates on sorted sets: it first obliviously merges the sorted input sets (complexity $O(n \log n)$ with a bitonic merger [Bat68]), then checks neighboring items for equality (complexity $O(n)$ with duplicate selector circuits [HEK12]), and finally obliviously shuffles the output to prevent leakage from the position of intersecting items (complexity $O(n \log n)$ with the Waksman shuffling network [Wak68]). Although the last step is not required when computing a symmetric analytics function on top of the intersection (where the result does not depend on the order of the inputs), the total complexity for this approach is $O(n \log n)$.

A line of work [PSZ14; PSSZ15; PSZ18] has further improved upon the SCS construction by introducing the concept of hashing items to bins first such that only the entries of corresponding bins must be compared as part of the oblivious circuit evaluation. Due to the possibility of hash collisions, each bin may contain more than a single item, more precisely up to $(\log n / \log \log n) \cdot (1 + o(1))$ items when hashing n items to n bins with a uniformly random hash function according to the analysis of [Gon81]. Since the information how many items are stored in a bin can in turn leak information about the items themselves, in a PSI protocol all bins must be padded to the maximum possible bin size with dummy items. To further reduce the number of required oblivious comparisons, the "PSZ" approach [PSZ14; PSSZ15; PSZ18] proposes to use a technique called Cuckoo hashing [PR01] for one of the two parties.

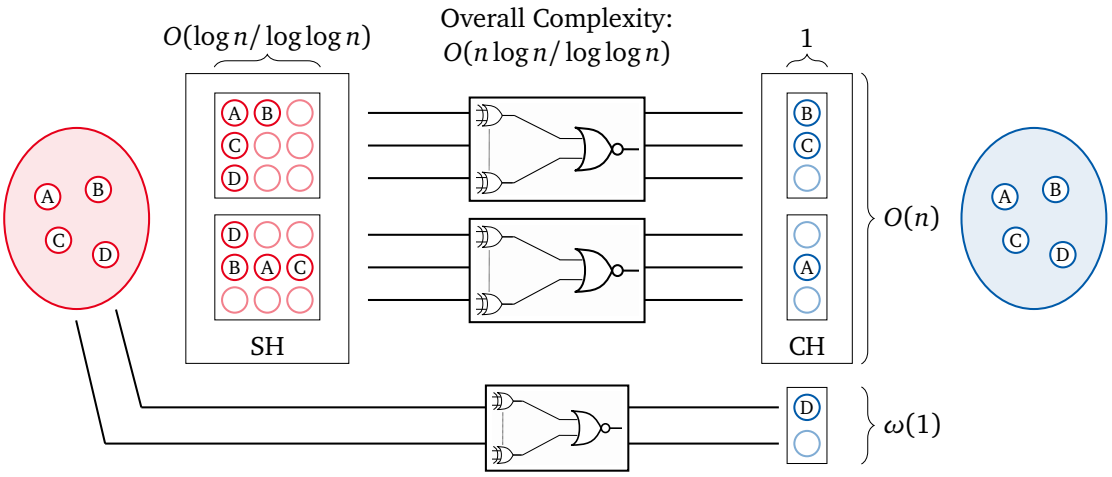


Figure 4.1: Circuit-based PSI based on simple hashing (SH) and Cuckoo hashing (CH) as proposed in [PSZ14; PSSZ15; PSZ18] with overall complexity $O(n \log n/\log\log n)$. For simplicity, the CH tables are split into separate tables for each hash function.

In Cuckoo hashing, each bin has a maximum capacity of 1 and an item is stored in one out of two possible bins determined by two different hash functions (there also exists 3-way Cuckoo hashing utilizing three hash functions, which we ignore for now). If a collision occurs during the insertion of an item, the currently stored item is evicted and placed in the bin whose position is determined by the respective other hash function. This procedure is repeated recursively, until no eviction is necessary or a recursion threshold is reached. In the latter case, the last evicted item is placed in a so-called *stash*. With a stash of size $s = \omega(1)$, this hashing scheme succeeds except with a negligible failure probability smaller than $n^{-(s-1)}$ for sufficiently large n [KMW08]. Since in Cuckoo hashing an item is stored in one out of two possible bins, the other party must prepare a hash table where each item is present in both locations. This corresponding scheme is called *simple hashing*. The circuit that computes the intersection then compares the single item in each Cuckoo bin with all items in the corresponding bin in the simple hashing table, and every item in the stash with every item contained in the other party's input set. The total complexity of this approach as visualized in Figure 4.1 is $O(n \log n/\log\log n)$.

This leaves the question: is it possible to construct circuit-based PSI with even lower complexity, e.g., with an asymptotically and ideally also concretely reduced number of comparisons that must be conducted obliviously? Finding an answer is important to increase the feasibility of conducting efficient privacy-preserving intersection analytics for large-scale databases.

4.1 Our Contributions

This thesis has contributed significantly to improving the asymptotic as well as concrete efficiency of circuit-based PSI and thus privacy-preserving database intersection analytics with the following publication that can be found in Appendix E:

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *37. Advances in Cryptology – EUROCRYPT'18*. Vol. 10822. LNCS. Code: https://encrypto.de/code/2DCH. Full version: https://ia.cr/2018/120. Springer, 2018, pp. 125–157. CORE Rank A*. Appendix E.

Specifically, we show for the first time that linear-complexity circuit-based PSI is in reach by giving a construction that performs only $\omega(n)$ oblivious comparisons and is concretely efficient. The core of our construction is a new hashing scheme that we call "2D Cuckoo hashing". Since the iterative version of our insertion algorithm is too complicated for formal analyses, we determine suitable parameters for our scheme to achieve negligible failure probability by conducting large-scale experiments on a high-performance computing cluster. Finally, in empirical performance evaluations, we demonstrate the concrete improvements in terms of computation and communication overhead over previous works [HEK12; PSZ14; PSSZ15; PSZ18].

2D Cuckoo Hashing. The goal in our work is to remove the $O(\log n/\log\log n)$ factor that stems from the requirement to pad the bins in regular simple hashing to the maximum possible extent in order to prevent information leakage during the PSI execution. For this, we split the simple hashing table into two tables T_L and T_R of size $O(n)$, each of which uses a different pair of hash functions and has a maximum bin size that is a small constant. Our insertion algorithm for what we call 2D Cuckoo hashing, upon detecting a collision in the left table that exceeds the defined maximum bin size, evicts the oldest present item from both positions in T_L and re-inserts the item in T_R. In case the insertion in T_R creates another collision such that the maximum bin size is exceeded, the oldest present item is removed from both positions in T_R and re-inserted in T_L. This procedure is iteratively repeated such that items are moved between T_L and T_R until no bin exceeds the maximum defined size. The intuition behind our iterative scheme is that elements that cause a high number of collisions in one table are evenly distributed in the respective other table when using a different pair of hash functions. The only question remaining is how to set the constant maximum bin size such that the failure probability of the scheme is negligible and the introduction of stashes can be avoided.

Since the simple hashing table is replaced with 2D Cuckoo hashing, we must adapt the hashing scheme of the second party that previously used regular Cuckoo hashing. For this, we suggest to use two regular Cuckoo hashing tables: the first table uses the same pair of hash functions as T_L in 2D Cuckoo hashing, and the second table uses the pair of hash functions known from T_R. Each item is then inserted in both Cuckoo tables. Thus, in the circuit that

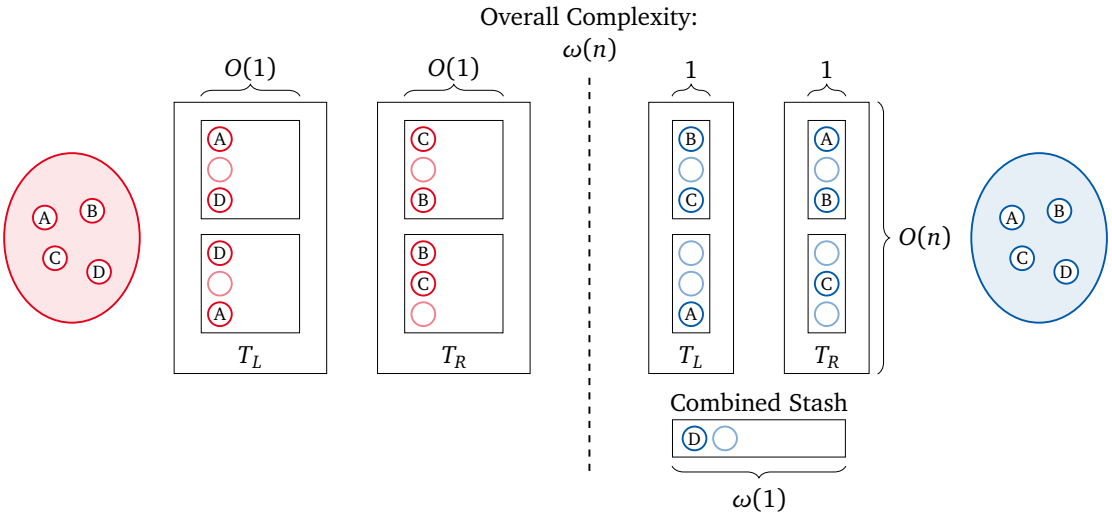


Figure 4.2: Circuit-based PSI based on 2D Cuckoo hashing as we propose in [**PSWW18**] with overall complexity $\omega(n)$. Again for simplicity, the hash tables are split into separate tables for each hash function. Also, comparison circuits between corresponding tables are omitted.

computes the intersection, each bin in the first Cuckoo table is compared to each item in the corresponding bin in T_L, and likewise each bin in the second Cuckoo table is compared to each item in the corresponding bin in T_R. This procedure guarantees that regardless of the placement of items on both sides, there is exactly one match when the item is contained in both parties' input sets.

Using two instances of Cuckoo hashing unfortunately entails two stashes. Each stash must be padded to the maximum possible size to not leak information in the PSI protocol and each item in each stash must be obliviously compared to each item in the other party's input set, which results in significant overhead. However, intuitively it is unlikely that both stashes are utilized to the maximum extent at the same time. Therefore, we suggest to use only a single combined stash. Based on the experimentally measured concrete stash sizes of [PSSZ15], we determine concrete stash sizes for the combined stash. It turns out that for large-scale input sets with more than a million items, the combined stash can even have the same size as a single stash. Thereby, we save 50 % of the stash-induced overhead, which helps to reduce the overall circuit size by a factor of 1.5× considering 32 bit inputs (cf. Figure 4.4).

We depict our 2D Cuckoo hashing construction with two instances of regular Cuckoo hashing and a combined stash in Figure 4.2. The two tables used in 2D Cuckoo hashing each have size $O(n)$ and both regular Cuckoo hashing tables have size $O(n)$. Thus, $O(n)$ comparisons between bins are required to determine the intersection. Additionally, each element in the combined stash of size $\omega(1)$ must be compared to n elements. Therefore, additional $\omega(n)$ comparisons are necessary. The overall complexity of our construction is therefore $\omega(n)$.

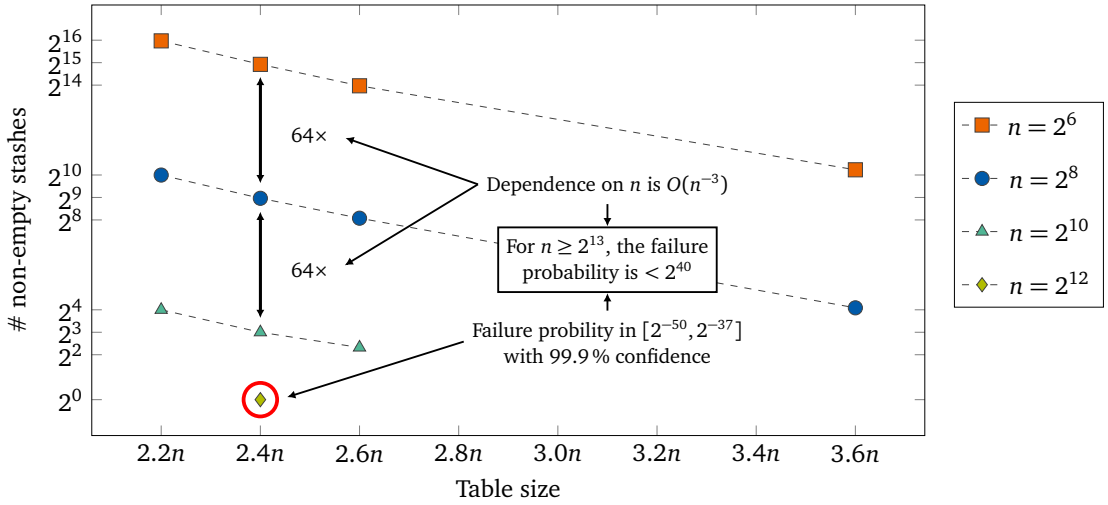


Figure 4.3: Simulation results for 2^{40} executions of 2D Cuckoo hashing on random inputs for variable table and set sizes.

Simulations for Setting Parameters. While there exist theoretic analyses for the performance of Cuckoo hashing [Wie17], we were unable to obtain similar results for 2D Cuckoo hashing with a bin size of 2 due to the complex nature of the iterative insertion algorithm. Instead, we provide empirical evidence that for sufficiently large input set sizes n the failure probability is negligible, i.e., below 2^{-40}. For this, we repeatedly run 2D Cuckoo hashing on random inputs for variable set, table, and bin sizes, and observe the number of cases where a stash would be required. Figure 4.3 summarizes our results when setting the maximum bin size in 2D Cuckoo hashing to 2 and performing 2^{40} experiments for $n \in \{2^6, 2^8, 2^{10}, 2^{12}\}$ for various table sizes. As we observe, the dependence of the failure probability on n is $O(n^{-3})$. From the concrete observations we can therefore state with high confidence that for $n \geq 2^{13}$ the failure probability for 2D Cuckoo hashing without a stash is indeed negligible, i.e., below 2^{-40}. All experiments required 5.5 M core hours on the Lichtenberg high-performance computer of the TU Darmstadt.

Performance Evaluation. We measure if the asymptotic improvement of our 2D Cuckoo hashing-based PSI construction over previous works also results in concrete performance improvements. For this, we first calculate the concrete circuit sizes in terms of non-free AND gates based on the number of oblivious comparisons that must be conducted. These results are depicted in Figure 4.4 and directly correlate with the concrete computation and communication overheads. For inputs of arbitrary lengths, we can observe a significant concrete performance improvement of factor 3.8× in communication over [PSSZ15] for $n = 2^{20}$.

We also implement our improved circuit-based PSI protocol based on the two-party MPC framework ABY [DSZ15] and provide run-time measurements using the GMW protocol [GMW87] (which already in [PSSZ15] outperformed Yao's garbled circuits protocol [Yao82; Yao86]) in Figure 4.5. All experiments are conducted on two machines equipped with an Intel

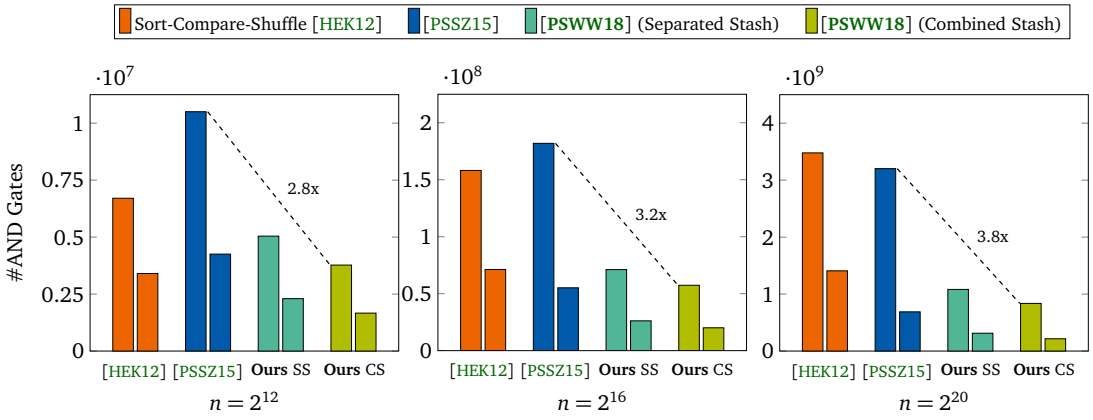


Figure 4.4: Comparison of concrete circuit sizes for circuit-based PSI protocols on inputs with arbitrary length (left) and 32 bit (right).

Core i7-4790 CPU clocked at 3.6 GHz and 16 GB of RAM connected in a LAN network with 1 Gbit/s bandwidth and 1 ms Round Trip Time (RTT). For simulating a WAN network, we restrict the bandwidth to 100 Mbit/s and set the RTT to 100 ms. For the circuit-based PSI protocols of [PSSZ15] and [**PSWW18**], we implement a lightweight PSI variant called "cardinality threshold" (PSI-CAT) that compares the cardinality of the intersection with a threshold and then outputs a single bit to indicate the comparison result. A modified PSI-CAT protocol that outputs the intersection if the threshold is exceeded has practical applications for privacy-preserving ride sharing [HOS17]. We compare these PSI-CAT implementations with the most efficient specialized PSI protocol at the time [KKRT16] (which outputs only the intersection) as well as the public-key-based PSI protocol of [Sha80; Mea86; CGT12] that outputs only the cardinality of the intersection (PSI-CA). Compared to [PSSZ15], we improve the total run-time by almost factor 3× in both the LAN and WAN setting for sets of size $n = 2^{20}$, and are two orders of magnitude faster than the public-key-based protocol of [Sha80; Mea86; CGT12].

4.2 Related Work

In the following, we review follow-up works that finally achieve two-party circuit-based PSI with $O(n)$ complexity, hybrid constructions, constructions for computing specific PSI variants, and further protocols to conduct privacy-preserving database intersection analytics.

The Road to Linear-Complexity Circuit-based PSI. Our work in [**PSWW18**] for circuit-based PSI with complexity $\omega(n)$ directly improves upon prior works with $O(n \log n)$ [HEK12] and $O(n \log n / \log \log n)$ complexity [PSZ14; PSSZ15; PSZ18] as described in §4 on Page 37. In recent years, follow-up works managed to achieve linear $O(n)$ complexity and further concrete performance improvements [PSTY19; CGS21; RS21]. We now briefly summarize

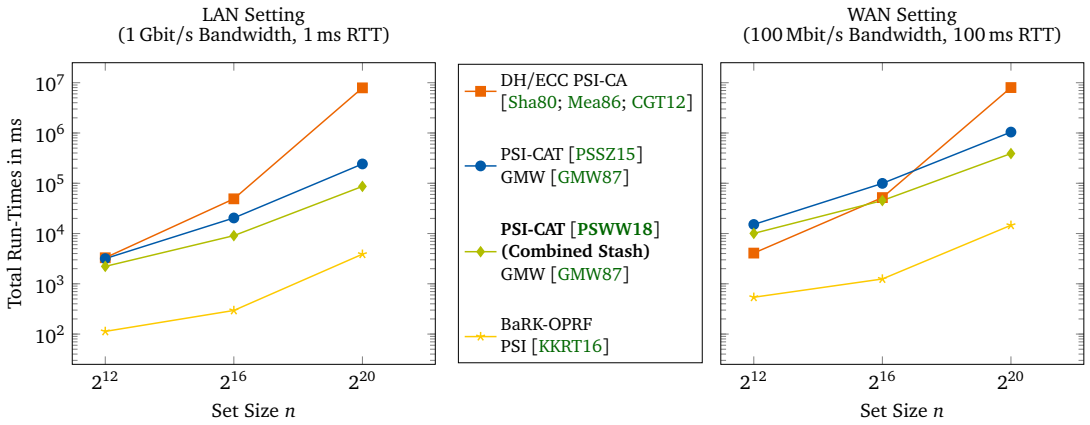


Figure 4.5: Comparison of total run-times for circuit-based PSI-CAT [PSSZ15; **PSWW18**] implemented in ABY [DSZ15] with specialized PSI [KKRT16] and public-key-based PSI-CA [Sha80; Mea86; CGT12].

the core ideas of [PSTY19; CGS21; RS21] and compare these constructions with prior works [HEK12; PSZ14; PSSZ15; PSZ18] and [**PSWW18**] in Table 4.1.

The basic setup in [PSTY19] is as in [PSZ14; PSSZ15; PSZ18] (cf. Figure 4.1), i.e., one party hashes their items via simple hashing and the other party via Cuckoo hashing. However, instead of directly performing comparisons between bin entries, they first have a pre-processing phase that runs an Oblivious Programmable Pseudo-Random Function (OPPRF) [KMP+17] between both parties. An OPPRF is an OPRF (cf. §2.1.2) that can be programmed to deliver a specifically defined output upon receiving a specific input. This can be utilized to program the same value for each item in a simple hashing bin. Therefore, the following circuit evaluation must only obliviously compare one element on each side instead of $O(\log n / \log\log n)$. By running a batched OPPRF over all bins, the bins are not required to be padded with dummy values to the maximum possible extent, which reduces to communication complexity of the OPPRF phase to $O(n)$. Additionally, by using 3-way Cuckoo hashing instead of 2-way Cuckoo hashing, the protocol of [PSTY19] can entirely avoid using a stash with super-constant size $\omega(1)$ [PSZ18]. (Note that the benefit of using 3-way Cuckoo hashing in the circuit-based protocols of [PSZ14; PSSZ15; **PSWW18**; PSZ18] would be diminished as then $3n$ instead of $2n$ items must be stored and compared in the simple hashing table.) The total communication complexity of [PSTY19] is therefore $O(n)$. For sets of size $n = 2^{20}$, the concrete communication overhead for arbitrary-length inputs is improved by a factor of 10.1× and the run-time by a factor of 2.8× and 5.8× compared to [**PSWW18**] in a LAN and a WAN network setting, respectively.

While the communication complexity of [PSTY19] is linear in n, the OPPRF phase incurs a super-linear computation overhead due to the employed interpolation method for large-degree polynomials. The work of [CGS21] reduces this computation overhead to be linear in n by introducing the notion of "relaxed batch" OPPRF (RB-OPPRF) evaluations, which

Protocol	Complexity		Hashing Scheme	Pre-Processing	Communication (in MB)
	Computation	Communication			
[HEK12]	$O(n \log n)$		—	—	106,144
[PSSZ15; PSZ18]	$O(n \log n / \log\log n)$		2-way CH + SH		97,708
[**PSWW18**]	$\omega(n)$		2D CH		25,532
[PSTY19]	$O(n(\log n)^2)$	$O(n)$	3-way CH + SH	OPPRF	2,540
[CGS21]	$O(n)$			RB-OPPRF	1,107
[RS21] (IKNP)				VOLE-based OPPRF	2,830
[RS21] (Silent OT)					277

Table 4.1: Comparison of two-party circuit-based PSI protocols. CH and SH denote Cuckoo and simple hashing, respectively. Three protocols before obliviously evaluating a circuit run interactive (Relaxed Batch (RB) / Vector Oblivious Linear Evaluation (VOLE)-based) Oblivious Programmable Pseudo-Random Function (OPPRF) protocols for pre-processing purposes. Communication is stated for input sets of size $n = 2^{20}$ with items of arbitrary length.

they efficiently construct based on Cuckoo hashing. In an RB-OPPRF evaluation, not one but three values are returned for each query such that exactly one of the returned values equals a programmed value when queried on the corresponding input. Whether one of the three values indeed is a value programmed for the entries in the simple hashing bins can be tested via Private Set Membership (PSM) protocols. The authors of [CGS21] provide two efficient PSM constructions (one OT-based and optimized for computation, the other OPPRF-based and optimized for communication) that output shares of a single bit to indicate the result. These shares are used as input for the following circuit-based computation. For the communication-efficient PSM protocol, [CGS21] improves communication over [PSTY19] for input sets of size $n = 2^{20}$ with arbitrary-length items by a factor of 2.3×; the run-time is improved by a factor of 2.7× and 2.1× in a LAN and a WAN network setting, respectively.

In concurrent and independent work to [CGS21], also [RS21] proposes a linear-complexity PSI protocol that can be utilized for circuit-based PSI. In their work, the authors replace the OPPRF of [PSTY19] with a construction based on a random-input Vector Oblivious Linear Evaluation (VOLE) protocol [SGRR19; WYKW20; YWL+20] with sublinear communication as well as a slightly modified version of the PaXoS data structure of [PRTY20] for compact encoding. For the subsequent circuit evaluation that performs equality checks on the OPPRF outputs, the authors of [RS21] experiment with replacing the OT extension protocol of [IKNP03] for the generation of multiplication triples in the precomputation phase of the GMW protocol [GMW87] with so-called "silent OT". Silent OT [BCG+19a; BCG+19b] is one of the most exciting recent breakthrough results in the area of secure computation. Based on the Learning Parity with Noise (LPN) assumption, it significantly reduces the communication overhead of OT extension protocols as parties can locally expand correlated seeds. The authors of [RS21] in an empirical performance evaluation study the trade-off between higher computation and reduced communication overhead. While the IKNP variant has 1.1× higher communication than [PSTY19] for input sets of size $n = 2^{20}$ with arbitrary-lengths items, the silent OT variant improves by factor 9.2× over [PSTY19]. In terms of run-time, in a LAN

and a WAN network, the IKNP variant outperforms [PSTY19] by a factor 3.1× and 1.2×, respectively. The highly communication-efficient silent OT variant in a LAN network is slower than [PSTY19] by a factor of 1.4×, but faster in a WAN network by a factor of 1.4×.

Further (Hybrid) Constructions for Circuit-based PSI. Besides the clear line of work towards linear-complexity circuit-based PSI outlined above, there have been other interesting works along the way that we discuss in the following.

In [FNO19], the authors propose to optimize the specialized PSI protocols of [PSZ14; PSSZ15; PSZ18], mainly by replacing the costly comparisons of stash elements with an OPRF-based unbalanced PSI protocol of [KLS+17]. Their construction for specialized PSI requires overall $O(n)$ communication, however, as an extension to generic PSI, they propose to use the sort-compare-shuffle approach of [HEK12] to run comparisons between corresponding bins. This results in a communication complexity of $O(n \log\log n)$, which does not beat concurrent works with $O(n)$ communication complexity [PSTY19]. For input sets of size $n = 2^{20}$ with arbitrary-length items, the approach of [FNO19] would require a concrete communication of 72,140 MB, compared to 25,532 MB and 2,540 MB for [**PSWW18**] and [PSTY19], respectively (cf. Table 4.1).

A notable innovation is the work of [CO18]. It is the first that suggests to combine the performance benefits of efficient specialized OT-based PSI protocols with the adaptability of circuit-based PSI. The challenge for such a hybrid construction is to provide the outputs of the equality tests in the first stage in "encrypted" form such that they can be directly used as input for MPC or HE without leaking any information about the intersection. According to this definition, also the constructions of [KK20; CGS21; GMR+21] are hybrid protocols. The protocol of [CO18] follows the Cuckoo/simple hashing paradigm known from the PSZ line of work [PSZ14; PSSZ15; PSZ18]. For each bin in the Cuckoo table, the idea is to perform a PSM protocol that uses OT to obliviously traverse a graph such that in the end a key is obtained to decrypt a share or wire label (depending on the subsequent MPC protocol) that encodes the PSM outcome. Due to the initial hashing setup, the complexity of the protocol is $O(n \log n / \log\log n)$ such that there is no asymptotic improvement over the prior works of [PSZ14; PSSZ15; PSZ18]. The lack of an implementation also prevents comparing concrete run-times. Potentially, our 2D Cuckoo hashing scheme of [**PSWW18**] could be applied to reduce the complexity down to $\omega(n)$.

The work of [KK20] presents a PSI protocol with $O(n)$ computation and communication complexity just as [CGS21; RS21]. The protocol of [KK20] replaces the OPPRF of [PSTY19] with a construction based on garbled Bloom filters [DCW13] and then performs equality tests outside a circuit using the protocol of [CO18]. However, their PSI protocol is not concretely efficient: for sets of size $n = 2^{16}$ (the maximum benchmarked set size) with 32 bit items, they require 4× more communication than [PSTY19] and their run-times in a LAN network are more than 10× higher.

The work of [GMR+21] achieves asymptotic and concrete improvements over [PSTY19] for sets with $\log n \ll l$ (where l is the bit length of the inputs) with the trade-off that the

cardinality of the intersection is inherently leaked. The core idea of [GMR+21] is to apply an oblivious shuffle step after the batched OPPRF phase of [PSTY19]. Then, the following equality checks are not part of an oblivious circuit evaluation but are carried out via the optimized protocol of [KKRT16] that outputs the result to one party in the clear. This leaks the cardinality of the intersection but not the intersection itself due to the previous shuffling step. With a further oblivious shuffling phase, it is possible to distribute shares of the intersection that can be used for further circuit-based computations. In terms of communication, for sets of size $n = 2^{20}$ with items of length 60 bit, the authors report an improvement of factor 2.6× over [PSTY19]. In terms of run-time, there is an improvement by factor 1.9× in a WAN network setting, however, due to the overhead of the switching network implementation in a LAN network, the run-time is 1.3× higher than for [PSTY19]. Overall, the use cases for this protocol are rather limited and potential application scenarios require careful analysis whether concrete performance gains can be expected and the cardinality leakage is acceptable.

Specific PSI Variants. Circuit-based PSI allows to compute arbitrary functions of the intersection. This flexibility naturally incurs significant overhead over specialized PSI protocols. However, there exist several specific PSI variants for practical applications that are in high demand. For these variants, specialized protocols have been devised with better performance than implementing the specific variant in any of the circuit-based PSI protocols described above. In the following, we highlight works that compute PSI-Cardinality, PSI-Sum, and further statistic operations.

Protocols for PSI-Cardinality output only the size of the intersection, but not the intersection itself, which has applications, e.g., in genome testing [BBC+11]. In circuit-based PSI protocols such as [**PSWW18**], this can be achieved by appending a circuit that computes the Hamming weight [BP06], i.e., counts the number of 1s in the vector that represents the output of the oblivious comparison step. However, there has been a long line of research proposing various constructions for this specific PSI variant [KS05; VC05; CZ09; BA12; CGT12; DD15; EFG+15]. Many of these works are simple variations of public-key-based PSI protocols [Sha80; Mea86]: by introducing an additional shuffle step, the receiving party can only determine that matches exist but not how they relate to their own inputs, thereby revealing only the cardinality of the intersection. These one-round protocols are very communication-efficient, however, may incur an unacceptable computation overhead for large set size, e.g., due to costly modular exponentiations. For example, for sets of size $n = 2^{20}$ with items of arbitrary length, the protocol of [CGT12] requires 66× less communication than [**PSWW18**] but has a 21× higher run-time even in a WAN network. Therefore, there has also been research that achieves accuracy/run-time trade-offs by providing only an approximate intersection cardinality [DL17].

PSI-Sum refers to protocols where one party has numerical payloads attached to their PSI inputs. The output of the protocol is only the sum of these payloads where the associated identifier is part of the intersection. In some use cases, also the size of the intersection should be revealed. The works of [IKN+17; IKN+20; MPR+20] studied so-called "PSI-Sum with cardinality" for measuring ad conversion rates in a privacy-preserving way. Here, the

goal is to find out how much money customers spent with a client company after seeing an advertisement on a platform of a service provider like Google. For this, in [IKN+17; IKN+20] the authors provide a protocol that computes the intersection with a simple DH-style protocol similar to [Mea86] and uses additively homomorphic Paillier encryption [Pai99] such that associated payloads for intersection items can be summed up. In [IKN+20], the authors also describe their considerations for deploying the protocol at Google, which represents one of the rare cases where MPC technology is actually used in practice at a large scale. For sets of size $n = 2^{20}$ with items of arbitrary length, the protocol of [IKN+17; IKN+20] requires 30× less communication but 6.4× more computation than implementing the same functionality in [PSTY19]. However, when applying concrete costs for bandwidth and core hours, the protocol of [IKN+17; IKN+20] turns out 868× cheaper. The later work of [MPR+20] adds security against malicious adversaries, which increases communication by about factor 4× and concrete monetary cost by factor 25×.

In [LPR+20], the authors present an extension to the PSI-Sum protocol of [IKN+20], which can be described as weighted PSI-Sum: the server for each item has additionally stored a weight such that the protocol obliviously computes the dot product with the values of matching items provided by the client. This allows the authors to implement a more fine-grained ad conversion rate measurement system that considers the time passed between viewing an ad and making a purchase as a weight to indicate the likelihood that the ad has influenced the buying decision. In light of the COVID-19 pandemic, this protocol also has applications to privacy-preserving contact tracing, where mobile clients check pseudo-random identifiers received within certain time intervals via Bluetooth against a database of reported cases that include a weight to indicate the severity of the infection. A notable contribution of [LPR+20] in this setting is the study of unbalanced set sizes, which is rare in circuit-based PSI since database intersection analytics are usually carried out only among service providers with rather symmetric set sizes. Specifically, similar to [KLS+17; RA18] and our work [**KRS+19**], the authors propose a protocol divided in setup and online phase, where the setup phase has linear complexity in the size of the server database and the online phase linear complexity in the size of the client database. Additionally, they propose a protocol based on Private Information Retrieval (PIR) [ACLS18; ALP+19] and Fully Homomorphic Encryption (FHE) [FV12; BGV14] that has a sublinear communication complexity in the size of the server database. For a server and client database of size $n = 2^{20}$ and 2^8, respectively, the first construction with offline setup requires 465 MB communication in the setup phase but only 7 MB in the online phase, compared to the circuit-based PSI protocol of [PSTY19] with 51 MB. The second construction with sublinear communication requires only 29 MB in total but has a run-time of 11 s per query in a LAN network, compared to amortized 0.18 s per query for [PSTY19] and 0.5 s/0.002 s for the setup/online construction.

With PSI-Stats, the authors of [YCP+20] essentially propose extensions for the protocol of [IKN+20] to compute various statistical functions (e.g., sum, percentile sum, mean, and standard deviation) on the payloads associated with intersecting elements. The authors empirically compare their protocol to [IKN+20] and [PSTY19]. Compared to the latter proto-

col computing only the intersection, PSI-Stats for computing the arithmetic mean requires about 4× less communication and slightly less run-time in a low-bandwidth network.

Database Intersection Analytics. In our work in [**PSWW18**], we contribute towards practical database intersection analytics via circuit-based two-party PSI. There have been recent works that allow for database intersection analytics in a broader sense using different computational and security models, which we discuss in the following.

The framework of [MRR20] allows two or more parties to perform composable SQL-like join queries on databases that are outsourced in secret-shared form to three servers assuming an honest majority. In this setting, it is possible to use very efficient three-party MPC protocols [AFL+16] that are known to have orders of magnitude higher throughput than two-party protocols. Using the framework to perform an inner join with a single column per database table equals the case of computing set intersection. One of the main challenges in the outsourcing scenario is to construct and utilize hashing schemes on secret-shared data to efficiently conduct comparisons. The authors of [MRR20] address this by constructing a shared Cuckoo hashing table of randomized encodings (created via LowMC [ARS+15], which we also utilized in §2.1.2) and using oblivious switching networks [MS13] to select the relevant locations for comparisons. In [MRR20], the authors furthermore describe the implementation of two concrete use cases (voter registration and threat log comparison) and also benchmark PSI operations. For sets of size $n = 2^{20}$, the work of [MRR20] reports about 25× faster run-times in both a LAN and a WAN setting, and about 5× less communication for computing PSI-Cardinality compared to our work in [**PSWW18**].

In [BKM+20], the authors address one inherent problem of conventional two-party circuit-based PSI protocols, namely that the entire computation must be repeated whenever one of the parties has a new data point that should be considered. With this background, they develop two DH-style protocols to compute some PSI variants. The first protocol, called "private ID" allows both parties to learn pseudo-random identifiers for each element in the set *union*, including a mapping of their own inputs to these IDs. Based on these IDs, the parties can locally sort their data sets such that any MPC protocol can efficiently compute aggregate functions over data associated with these IDs. The second protocol is called "private secret-shared set intersection". It outputs additive shares generated via additively homomorphic Paillier encryption [Pai99] of values that are associated with matching identifiers. Here, only one party must initially have the entire set ready, whereas the other party can query batches in a streaming fashion. Again, arbitrary MPC protocols can be utilized to compute aggregate functions over the shared data pairs afterwards.

Privacy-preserving database intersection analytics can be implemented in any framework for secure two- or multi-party computation that can be programmed via a high-level or query language as well as circuit-level specification (see [HHNZ19] for a detailed survey). However, if not paired with optimized hashing schemes as we present with 2D Cuckoo hashing [**PSWW18**], straight-forward implementations in these frameworks likely result in the naive $O(n^2)$ complexity approach outlined in §4 on Page 37. Exceptions are frameworks that include query optimizers for database joins and aggregations such as SMCQL [BEE+17],

Conclave [VSG+19], and Senate [PKY+21]. Senate specifically includes optimized circuit building blocks for multi-way set intersections that are instantiated for according join queries. While there also exist works that provide differentially private database analytics (e.g., [NH12; JNS17; PNH17; JNS18; KKL+20]), we focus on exact methods, which are required, for example, in the medical domain to perform precise analyses of genomic data [BBC+11; **TWSH18**].

5 Conclusion and Future Work

In this final chapter, we first summarize the results presented in this thesis (cf. §5.1) and then discuss important directions of future work (cf. §5.2).

5.1 Summary

In this thesis, we studied how to make Private Set Intersection (PSI) protocols efficient enough to design practical privacy-preserving alternatives for three important and widely used real-world applications: mobile contact discovery, mutual authentication for Apple AirDrop, and database intersection analytics. The need for such alternatives is warranted by attacks that we demonstrated for currently deployed systems, which showed that users' privacy is severely threatened at the moment. In the following, we briefly summarize our contributions to the three studied application scenarios.

Mobile Contact Discovery. In [HWS+21], we conducted large-scale crawling attacks on three popular mobile messengers (WhatsApp, Telegram, and Signal) that exploit insufficient rate limits in the service providers' contact discovery APIs. We systematically compared the amount and type of leaked personal information, reported interesting (cross-messenger) usage statistics, and documented the service providers' (insufficient) protection measures. We proposed suitable countermeasures to prevent enumeration attacks, most notably a PSI-compatible incremental contact discovery scheme that strictly improves over Signal's current approach while not storing synchronization state information on the server side. Furthermore, we showed that naive hashing-based protocols are severely insufficient to provide privacy-preserving contact discovery. For this, we studied and compared three hash reversal techniques: large-scale in-memory databases with a Redis cluster, brute-force attacks with hashcat, and a rainbow table construction for non-uniform input domains that can reverse any mobile phone number hash in the order of milliseconds. As a privacy-preserving alternative, in [KRS+19], we proposed two unbalanced PSI protocols with security against malicious clients that scale to large user database sizes due to our optimizations in terms of Cuckoo filter compression and updates, specialized pseudo-random functions, and hardware acceleration of cryptographic operations on modern ARMv8-A smartphone SoCs. We integrated our PSI protocol implementations into the open-source platform CogniCrypt [KNR+17] to make them readily available to developers. All our results are summarized at https://contact-discovery.github.io, including links to an up-to-date

press review and explainer videos to inform the general public about our research activities and raise awareness for existing privacy vulnerabilities.

Mutual Authentication for Apple AirDrop. Based on previous reverse-engineering efforts of Apple's proprietary AirDrop protocol, we discovered two privacy vulnerabilities that leak the (hashed) contact identifiers associated with users' Apple IDs during the AirDrop authentication handshake. In [**HHS+21a**], we demonstrated the practicality of both attacks with "AirCollect" that extends our specialized rainbow table construction from [**HWS+21**] with support for SHA-256 hashing to immediately reverse mobile phone number hashes collected from nearby Apple users. As there is no trivial fix for the existing AirDrop protocol, in [**HHS+21b**] we therefore proposed "PrivateDrop", a privacy-preserving version of AirDrop based on two consecutive executions of an optimized maliciously-secure PSI protocol to provide a mutual contact identifier-based authentication mechanism. We implemented our solution in Apple's native programming language Swift and demonstrated its practicality on iPhones and MacBooks with an authentication delay that is below one second and therefore perceived as an immediate response by users. A summary of the impact of our attacks and proposed solution can be found at `https://privatedrop.github.io`.

Database Intersection Analytics. As two-party circuit-based PSI protocols are a promising approach to realize privacy-preserving database intersection analytics, in [**PSWW18**] we proposed a new construction that asymptotically and concretely improved over all prior works. The core of our protocol is a novel hashing scheme called "2D Cuckoo hashing" for which we determined fail-safe parameters by spending 5.5 million core hours on the high-performance computer of TU Darmstadt. Our PSI protocol has $\omega(n)$ complexity and our implementation for a cardinality-threshold variant in the ABY two-party MPC framework [DSZ15] performed 3× faster in terms of total run-time for databases with a million entries each than the best prior works of [PSSZ15; PSZ18] in both a LAN and WAN network setting. While follow-up works such as [PSTY19; CGS21; RS21] are strict improvements over [**PSWW18**], we made an important step in bringing the research field close to achieving truly linear $O(n)$ computation and communication complexity.

5.2 Future Work

In the following, we propose potential directions of future work for each of the investigated applications: mobile contact discovery in §5.2.1, mutual authentication for Apple AirDrop in §5.2.2, and database intersection analytics in §5.2.3.

5.2.1 Mobile Contact Discovery

For improved privacy in mobile contact discovery, we suggest as part of future work to analyze the implementations of further mobile applications and services with a contact importer

feature, to design and implement an ideal functionality specifically for contact discovery, and to investigate combinations of multi-server PIR with Intel SGX.

Analysis of Contact Discovery Implementation in Further Applications. In [HWS+21], we investigated the contact discovery implementations of three of the world's most popular mobile messengers. Interestingly, even a service operated by a leading technology company such as Facebook turned out to have severely insufficient measures in place to protect users' personal data. Therefore, it would be appropriate to broaden the scope to investigate further messengers and other mobile applications that offer a contact discovery feature, e.g., the video chat-based social network application Clubhouse[1]. Since this is a tedious task when done manually (as with our crawling setup), it would be helpful to create an automation framework that requires only minimal configuration to be adapted to new applications. Such a framework could automate the task of populating address books in an Android emulator, triggering the contact discovery procedure of an application, and take note of the outcome. Furthermore, it would be interesting to broaden the scope outside of mobile applications and look at other platforms that offer contact importer features. For example, the professional networking website LinkedIn[2] offers its users to automatically populate the network of existing business contacts – if users provide their login credentials for their (professional) email account. Of course, given the confidential nature of private or professional email exchange, such a contact discovery should be replaced with a privacy-preserving alternative, for example with a PSI protocol execution run between the email provider and LinkedIn.

Ideal Functionality for Contact Discovery. In both [KRS+19] and [HWS+21] we advocated to use PSI protocols as a measure to provide privacy-preserving contact discovery. At the same time, in [HWS+21], we criticized insufficient rate limits applied by Signal, a service that does not store synchronization state information on the server side. It is important to note that likewise the use of PSI prevents enforcing stricter rate limits as for the service provider there is no way to check to which degree the client's input set has changed between requests, except for matching contacts. This leads to the question whether the ideal functionality of set intersection is well-suited for being applied to contact discovery. Instead, it would be intriguing to design an ideal functionality specifically for contact discovery that can enforce limits on the number of changes to the client input set between sessions. Such a functionality then in general could also be used to realize PSI with built-in rate limits. An even bigger challenge, however, is to design an efficient cryptographic protocols to securely realize such a functionality.

Combination of multi-server PIR with Intel SGX. In [KRS+19], we proposed the combination of our PSI protocols with multi-server PIR [BGI16] in order to reduce the setup costs, which for extremely large user databases are still impractical and consist of transferring multiple gigabytes of data. However, service providers like Signal have expressed concerns doubting the existence of non-colluding servers (which are required for multi-server PIR)

[1] https://www.joinclubhouse.com
[2] https://www.linkedin.com

and instead pursue privacy-preserving contact discovery purely based on Intel's TEE implementation SGX. For our proposed combination of PSI and PIR to nevertheless gain traction, the following steps are required: First, it would be necessary to determine the concrete communication costs for multi-server PIR [BGI16] for the use case of contact discovery and if promising, come up with a prototype implementation to assess the performance overhead in terms of runtime. Then, it would be interesting to explore combinations of such multi-server PIR protocols with Intel SGX, which in this context would not primarily be used to protect the confidentiality of data, but to effectively prevent collusion between the servers, as the enclaves via remote attestation can ensure the client applications of the integrity of the code that processes PIR queries. Colluding service providers then both would need to run sophisticated side-channel attacks to reconstruct the users' queries from observing the enclave execution.

5.2.2 Mutual Authentication for Apple AirDrop

To further increase privacy for AirDrop, we suggest to prevent user tracking via TLS certificates. Additionally, we encourage to extend the study of contact-based mutual authentication protocols beyond the Apple ecosystem by investigating the authentication mechanisms in Google's new Nearby platform.

User Tracking via TLS Certificates. In our work [**HHS+21b**], we proposed a mutual authentication protocol that prevents users from unintentionally leaking their contact identifiers. The messages of our protocol (as well as of the original AirDrop protocol) are sent over a TLS connection that secures the Apple Wireless Direct Link (AWDL) traffic from passive eavesdroppers. Unfortunately, the certificates that are used to establish the TLS connection contain a static and account-specific Universally Unique Identifier (UUID). While this does not allow to directly map a user's identity to a specific location, it nevertheless allows for tracking. For example, by deploying a large number of cheap Wi-Fi-enabled devices throughout a city, it is possible to build accurate movement profiles simply by triggering a connection and extracting the UUID from the certificate. For powerful adversaries who have access to surveillance cameras in public hot spots and can run facial recognition software, it might even be possible to map UUIDs to identities to closely monitor the behavior of citizens. Thus, preventing user tracking via TLS certificates is the next important step in order to offer a privacy-preserving AirDrop service. A simple solution would be for Apple to issue a larger number of certificates (e.g., 100) with different UUIDs such that devices for each new connection can randomly choose one. This approach does not entirely prevent user tracking but makes movement profiles significantly more coarse-grained.

Google Nearby. With PrivateDrop, we design in [**HHS+21b**] a privacy-preserving solution specifically for Apple's AirDrop protocol. However, the underlying principle of two consecutive PSI protocol executions can be transferred outside the Apple ecosystem for all services that base authentication on the notion of users being mutual contacts. One such

service is Google's recently introduced alternative for the Android ecosystem called "Nearby"[3]. Similar to AirDrop, Nearby offers users the option to restrict device visibility to specific known contacts [Sch20]. Thus, as part of future work, it would be interesting to closely examine how Nearby currently implements the authentication handshake between devices. In case there are similar issues present as in Apple's current insecure AirDrop implementation, our PrivateDrop protocol could be adapted.

5.2.3 Database Intersection Analytics

For more developer-friendly, communication-efficient, and versatile database intersection analytics, we propose as part of future work in the area of circuit-based PSI to integrate specialized query compilers and to extend the protocols to the multi-party case.

Query Compilers for Circuit-based PSI. Proposals for circuit-based PSI, including our work [**PSWW18**], are only concerned about designing efficient protocols for computing the intersection such that the result is not directly revealed but available in a subsequent secure circuit evaluation. However, how to efficiently implement the actual analytics functionality in such a circuit is left to the developer. While there exist specialized circuit compilers that compile high-level languages such as C/C++ to optimized Boolean or arithmetic circuits (e.g., [BDK+18; **HST+21**]), for database analytics it would be more convenient for developers to program in an SQL-like query language. Such query compilers already exist in other privacy-preserving database analytics frameworks, for example, [BEE+17; VSG+19; MRR20; PKY+21]. Therefore, it would be interesting to integrate such query compilers with optimized circuit-based PSI protocols such as [**PSWW18**; PSTY19; CGS21; RS21].

Multi-Party Circuit-based PSI. In our work [**PSWW18**], we only studied circuit-based PSI protocols in the two-party setting. In terms of multi-party circuit-based PSI, the only concretely efficient protocol with linear complexity was very recently proposed in [CDG+21]. However, this protocol operates in a setting that assumes an honest majority of protocol participants, which might not always be reasonable. Therefore, as part of future work, it would be interesting to construct efficient circuit-based multi-party PSI in the dishonest majority or even full-threshold setting, where all but one parties might be corrupted. Additionally, for an extremely large number of protocol participants, it would be interesting to study efficient outsourcing to a smaller number of servers. Here, the challenge is to use a hashing scheme on secret-shared inputs or otherwise construct an efficient protocol.

[3] https://developers.google.com/nearby

Bibliography

[Aba02] M. ABADI. **"Private Authentication"**. In: *Workshop on Privacy Enhancing Technologies (PET)*. Vol. 2482. LNCS. Springer, 2002, pp. 27–40.

[ACLS18] S. ANGEL, H. CHEN, K. LAINE, S. T. V. SETTY. **"PIR with Compressed Queries and Amortized Query Processing"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2018, pp. 962–979.

[AF04] M. ABADI, C. FOURNET. **"Private Authentication"**. In: *Theoretical Computer Science* 322.3 (2004). Elsevier, pp. 427–476.

[AFL+16] T. ARAKI, J. FURUKAWA, Y. LINDELL, A. NOF, K. OHARA. **"High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 805–817.

[AG20] J. AAS, T. GEOGHEGAN. **"Introducing ISRG Prio Services for Privacy Respecting Metrics"**. 2020. URL: https://www.abetterinternet.org/post/introducing-prio-services/ (visited on 07/02/2021).

[AGM+18] D. F. ARANHA, C. P. L. GOUVÊA, T. MARKMANN, R. S. WAHBY, K. LIAO. **"RELIC is an Efficient LIbrary for Cryptography"**. 2018. URL: https://github.com/relic-toolkit/relic (visited on 07/02/2021).

[ALP+19] A. ALI, T. LEPOINT, S. PATEL, M. RAYKOVA, P. SCHOPPMANN, K. SETH, K. YEO. **"Communication-Computation Trade-offs in PIR"**. In: *Cryptology ePrint Archive* 2019/1483 (2019). IACR.

[ALSZ13] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. **"More Efficient Oblivious Transfer and Extensions for Faster Secure Computation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 535–548.

[ARS+15] M. R. ALBRECHT, C. RECHBERGER, T. SCHNEIDER, T. TIESSEN, M. ZOHNER. **"Ciphers for MPC and FHE"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 9056. LNCS. Springer, 2015, pp. 430–454.

[BA12] M. BLANTON, E. AGUIAR. **"Private and Oblivious Set and Multiset Operations"**. In: *ACM Asia Conference on Computer and Communication Security (AsiaCCS)*. ACM, 2012, pp. 40–41.

[Bat68] K. E. BATCHER. **"Sorting Networks and Their Applications"**. In: *AFIPS Spring Joint Computing Conference*. Vol. 32. AFIPS Conference Proceedings. Thomson Book Company, Washington D.C., 1968, pp. 307–314.

[BBC+11] P. BALDI, R. BARONIO, E. D. CRISTOFARO, P. GASTI, G. TSUDIK. **"Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2011, pp. 691–702.

[BCG+19a] E. BOYLE, G. COUTEAU, N. GILBOA, Y. ISHAI, L. KOHL, P. RINDAL, P. SCHOLL. **"Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 291–308.

[BCG+19b] E. BOYLE, G. COUTEAU, N. GILBOA, Y. ISHAI, L. KOHL, P. SCHOLL. **"Efficient Pseudorandom Correlation Generators: Silent OT Extension and More"**. In: *Advances in Cryptology – CRYPTO*. Vol. 11694. LNCS. Springer, 2019, pp. 489–518.

[BDK+18] N. BÜSCHER, D. DEMMLER, S. KATZENBEISSER, D. KRETZMER, T. SCHNEIDER. **"HyCC: Compilation of Hybrid Protocols for Practical Secure Computation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 847–861.

[BEE+17] J. BATER, G. ELLIOTT, C. EGGEN, S. GOEL, A. N. KHO, J. ROGERS. **"SMCQL: Secure Query Processing for Private Data Networks"**. In: *Proceedings of the VLDB Endowment* 10.6 (2017). VLDB Endowment, pp. 673–684.

[BF01] D. BONEH, M. K. FRANKLIN. **"Identity-Based Encryption from the Weil Pairing"**. In: *Advances in Cryptology – CRYPTO*. Vol. 2139. LNCS. Springer, 2001, pp. 213–229.

[BGI16] E. BOYLE, N. GILBOA, Y. ISHAI. **"Function Secret Sharing: Improvements and Extensions"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 1292–1303.

[BGK08] A. BOLDYREVA, V. GOYAL, V. KUMAR. **"Identity-Based Encryption with Efficient Revocation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2008, pp. 417–426.

[BGV14] Z. BRAKERSKI, C. GENTRY, V. VAIKUNTANATHAN. **"(Leveled) Fully Homomorphic Encryption without Bootstrapping"**. In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014). ACM, 13:1–13:36.

[BHKR13] M. BELLARE, V. T. HOANG, S. KEELVEEDHI, P. ROGAWAY. **"Efficient Garbling from a Fixed-Key Blockcipher"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2013, pp. 478–492.

[BJ18] A. D. BRESLOW, N. JAYASENA. **"Morton Filters: Faster, Space-Efficient Cuckoo Filters via Biasing, Compression, and Decoupled Logical Sparsity"**. In: *Proceedings of the VLDB Endowment* 11.9 (2018). VLDB Endowment, pp. 1041–1055.

[BKM+20] P. BUDDHAVARAPU, A. KNOX, P. MOHASSEL, S. SENGUPTA, E. TAUBENECK, V. VLASKIN. **"Private Matching for Compute"**. In: *Cryptology ePrint Archive* 2020/599 (2020). IACR.

[BKN+14] A. BUCHENSCHEIT, B. KÖNINGS, A. NEUBERT, F. SCHAUB, M. SCHNEIDER, F. KARGL. **"Privacy Implications of Presence Sharing in Mobile Messaging Applications"**. In: *Mobile and Ubiquitous Multimedia (MUM)*. ACM, 2014, pp. 20–29.

[Blo70] B. H. BLOOM. **"Space/Time Trade-offs in Hash Coding with Allowable Errors"**. In: *Communications of the ACM (CACM)* 13.7 (1970). ACM, pp. 422–426.

[BMS+20] J. V. BULCK, D. MOGHIMI, M. SCHWARZ, M. LIPP, M. MINKIN, D. GENKIN, Y. YAROM, B. SUNAR, D. GRUSS, F. PIESSENS. **"LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2020, pp. 54–72.

[BMW+18] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, R. Strackx. **"Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution"**. In: *USENIX Security Symposium*. USENIX Association, 2018, pp. 991–1008.

[BNP08] A. Ben-David, N. Nisan, B. Pinkas. **"FairplayMP: a system for secure multi-party computation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2008, pp. 257–266.

[BNPS03] M. Bellare, C. Namprempre, D. Pointcheval, M. Semanko. **"The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme"**. In: *Journal of Cryptology (JoC)* 16.3 (2003). Springer, pp. 185–215.

[Böc21] H. Böck. **"Signal-Server nicht mehr Open Source"**. 2021. URL: https://www.golem.de/news/crypto-messenger-signal-server-nicht-mehr-open-source-2104-155376.html (visited on 07/02/2021).

[BP06] J. Boyar, R. Peralta. **"Concrete Multiplicative Complexity of Symmetric Functions"**. In: *Mathematical Foundations of Computer Science (MFCS)*. Vol. 4162. LNCS. Springer, 2006, pp. 179–189.

[BPH+10] M. Balduzzi, C. Platzer, T. Holz, E. Kirda, D. Balzarotti, C. Kruegel. **"Abusing Social Networks for Automated User Profiling"**. In: *Recent Advances in Intrusion Detection (RAID)*. Vol. 6307. LNCS. Springer, 2010, pp. 422–441.

[BSBK09] L. Bilge, T. Strufe, D. Balzarotti, E. Kirda. **"All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks"**. In: *World Wide Web (WWW)*. ACM, 2009, pp. 551–560.

[Can21a] K. Canales. **"Hackers Scraped Data From 500 Million LinkedIn Users"**. 2021. URL: https://www.businessinsider.com/linkedin-data-scraped-500-million-users-for-sale-online-2021-4 (visited on 07/02/2021).

[Can21b] K. Canales. **"Scraped Personal Data of 1.3 Million Clubhouse Users Has Reportedly Been Posted Online"**. 2021. URL: https://www.businessinsider.com/clubhouse-data-leak-1-million-users-2021-4 (visited on 07/02/2021).

[CB17] H. Corrigan-Gibbs, D. Boneh. **"Prio: Private, Robust, and Scalable Computation of Aggregate Statistics"**. In: *Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2017, pp. 259–282.

[CDG+21] N. Chandran, N. Dasgupta, D. Gupta, S. L. B. Obbattu, S. Sekar, A. Shah. **"Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI"**. In: *Cryptology ePrint Archive* 2021/172 (2021). IACR.

[CGS21] N. Chandran, D. Gupta, A. Shah. **"Circuit-PSI with Linear Complexity via Relaxed Batch OPPRF"**. In: *Cryptology ePrint Archive* 2021/34 (2021). IACR.

[CGT12] E. D. Cristofaro, P. Gasti, G. Tsudik. **"Fast and Private Computation of Cardinality of Set Intersection and Union"**. In: *Cryptology And Network Security (CANS)*. Vol. 7712. Springer, 2012, pp. 218–231.

[Cha19] D. Chastuhin. **"Apple Bleee: Everyone Knows What Happens on Your iPhone"**. 2019. URL: https://hexway.io/research/apple-bleee/ (visited on 07/02/2021).

[CHLR18] H. Chen, Z. Huang, K. Laine, P. Rindal. **"Labeled PSI from Fully Homomorphic Encryption with Malicious Security"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 1223–1237.

[CKT10] E. D. CRISTOFARO, J. KIM, G. TSUDIK. **"Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model"**. In: *Advances in Cryptology – ASIACRYPT*. Vol. 6477. LNCS. Springer, 2010, pp. 213–231.

[CLR17] H. CHEN, K. LAINE, P. RINDAL. **"Fast Private Set Intersection from Homomorphic Encryption"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 1243–1255.

[CNKX15] A. CHATZIKONSTANTINOU, C. NTANTOGIAN, G. KAROPOULOS, C. XENAKIS. **"Evaluation of Cryptography Usage in Android Applications"**. In: *Bio-inspired Information and Communications Technologie (BICT)*. ICST/ACM, 2015, pp. 83–90.

[CO13] K. CHURCH, R. d. OLIVEIRA. **"What's up with WhatsApp?: Comparing Mobile Instant Messaging Behaviors with Traditional SMS"**. In: *Human-Computer Interaction with Mobile Devices and Services (MobileHCI)*. ACM, 2013, pp. 352–361.

[CO15] T. CHOU, C. ORLANDI. **"The Simplest Protocol for Oblivious Transfer"**. In: *Advances in Cryptology – LATINCRYPT*. Vol. 9230. LNCS. Springer, 2015, pp. 40–58.

[CO18] M. CIAMPI, C. ORLANDI. **"Combining Private Set-Intersection with Secure Two-Party Computation"**. In: *Security and Cryptography for Networks (SCN)*. Vol. 11035. LNCS. Springer, 2018, pp. 464–482.

[Con18] N. CONFESSORE. **"Cambridge Analytica and Facebook: The Scandal and the Fallout So Far"**. 2018. URL: https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html (visited on 07/02/2021).

[Cox17] J. COX. **"Building a Database of WhatsApp Users Can Be Pretty Easy"**. 2017. URL: https://www.vice.com/en/article/wnw4vw/building-a-database-of-whatsapp-users-can-be-pretty-easy (visited on 07/02/2021).

[CRM91] S. K. CARD, G. G. ROBERTSON, J. D. MACKINLAY. **"The Information Visualizer, an Information Workspace"**. In: *Conference on Human Factors in Computing Systems (CHI)*. ACM, 1991, pp. 181–186.

[CT10] E. D. CRISTOFARO, G. TSUDIK. **"Practical Private Set Intersection Protocols with Linear Complexity"**. In: *Financial Cryptography and Data Security (FC)*. Vol. 6052. LNCS. Springer, 2010, pp. 143–159.

[CT12] E. D. CRISTOFARO, G. TSUDIK. **"Experimenting with Fast Private Set Intersection"**. In: *Trust and Trustworthy Computing (TRUST)*. Vol. 7344. LNCS. Springer, 2012, pp. 55–73.

[CYJ+13] Y. CHENG, L. YING, S. JIAO, P. SU, D. FENG. **"Bind Your Phone Number with Caution: Automated User Profiling through Address Book Matching on Smartphone"**. In: *ACM Asia Conference on Computer and Communication Security (AsiaCCS)*. ACM, 2013, pp. 335–340.

[CZ09] J. CAMENISCH, G. M. ZAVERUCHA. **"Private Intersection of Certified Sets"**. In: *Financial Cryptography and Data Security (FC)*. Vol. 5628. LNCS. Springer, 2009, pp. 108–127.

[Dat19] DATAFINDER. **"Recover Encrypted Email Addresses"**. 2019. URL: https://web.archive.org/web/20191211152224/https://datafinder.com/products/email-recovery (visited on 07/02/2021).

[DCW13] C. DONG, L. CHEN, Z. WEN. **"When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 789–800.

[DD15] S. K. DEBNATH, R. DUTTA. **"Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter"**. In: *Information Security Conference (ISC)*. Vol. 9290. LNCS. Springer, 2015, pp. 209–226.

[DKCL18] L. DEMIR, A. KUMAR, M. CUNCHE, C. LAURADOUX. **"The Pitfalls of Hashing for Privacy"**. In: *IEEE Communications Surveys and Tutorials* 20.1 (2018). IEEE Computer Society, pp. 551–565.

[DKLS18] J. DOERNER, Y. KONDI, E. LEE, A. SHELAT. **"Secure Two-party Threshold ECDSA from ECDSA Assumptions"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2018, pp. 980–997.

[DKS+21] D. DEMMLER, S. KATZENBEISSER, T. SCHNEIDER, T. SCHUSTER, C. WEINERT. **"Improved Circuit Compilation for Hybrid MPC via Compiler Intermediate Representation"**. In: *Security and Cryptography (SECRYPT)*. SciTePress, 2021, pp. 444–451.

[DL17] C. DONG, G. LOUKIDES. **"Approximating Private Set Union/Intersection Cardinality With Logarithmic Complexity"**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 12.11 (2017). IEEE Computer Society, pp. 2792–2806.

[DMRY09] D. DACHMAN-SOLED, T. MALKIN, M. RAYKOVA, M. YUNG. **"Efficient Robust Private Set Intersection"**. In: *Applied Cryptography and Network Security (ACNS)*. Vol. 5536. LNCS. 2009, pp. 125–142.

[Dof19] Z. DOFFMAN. **"Instagram Confirms Security Issue Exposed User Accounts And Phone Numbers"**. 2019. URL: https://www.forbes.com/sites/zakdoffman/2019/09/12/new-instagram-hack-exclusive-facebook-confirms-user-accounts-and-phone-numbers-at-risk/ (visited on 07/02/2021).

[Dom20] DOMO, INC. **"Data Never Sleeps 8.0"**. 2020. URL: https://www.domo.com/learn/data-never-sleeps-8 (visited on 07/02/2021).

[DRRT18] D. DEMMLER, P. RINDAL, M. ROSULEK, N. TRIEU. **"PIR-PSI: Scaling Private Contact Discovery"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2018.4 (2018). De Gruyter Open, pp. 159–178.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015.

[EBFK13] M. EGELE, D. BRUMLEY, Y. FRATANTONIO, C. KRUEGEL. **"An Empirical Study of Cryptographic Misuse in Android Applications"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 73–84.

[EFG+15] R. EGERT, M. FISCHLIN, D. GENS, S. JACOB, M. SENKER, J. TILLMANNS. **"Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters"**. In: *Australasian Conference on Information Security (ACISP)*. Vol. 9144. LNCS. Springer, 2015, pp. 413–430.

[Eur16] EUROPEAN PARLIAMENT, COUNCIL OF THE EUROPEAN UNION. **"Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation)"**. In: *Official Journal of the European Union* L.119 (2016). European Union, pp. 1–88.

[FAKM14] B. FAN, D. G. ANDERSEN, M. KAMINSKY, M. MITZENMACHER. **"Cuckoo Filter: Practically Better Than Bloom"**. In: *Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2014, pp. 75–88.

[FHNP16] M. J. FREEDMAN, C. HAZAY, K. NISSIM, B. PINKAS. **"Efficient Set Intersection with Simulation-Based Security"**. In: *Journal of Cryptology (JoC)* 29.1 (2016). Springer, pp. 115–155.

[FNO19] B. H. FALK, D. NOBLE, R. OSTROVSKY. **"Private Set Intersection with Linear Communication from General Assumptions"**. In: *Workshop on Privacy in the Electronic Society (WPES@CCS)*. ACM, 2019, pp. 14–25.

[FNP04] M. J. FREEDMAN, K. NISSIM, B. PINKAS. **"Efficient Private Matching and Set Intersection"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 3027. LNCS. Springer, 2004, pp. 1–19.

[Fra21] L. FRANCESCHI-BICCHIERAI. **"You Can't Hide Whether You Are Online on WhatsApp"**. 2021. URL: https://maikel.pro/blog/whatsspy-public-support-ending-today (visited on 07/02/2021).

[FV12] J. FAN, F. VERCAUTEREN. **"Somewhat Practical Fully Homomorphic Encryption"**. In: *Cryptology ePrint Archive* 2012/144 (2012). IACR.

[GGAK16] S. GUPTA, P. GUPTA, M. AHAMAD, P. KUMARAGURU. **"Exploiting Phone Numbers and Cross-Application Features in Targeted Mobile Attacks"**. In: *Security and Privacy in Smartphones (SPSM@CCS)*. ACM, 2016, pp. 73–82.

[Gla17] G. GLAUDELL. **"Digital Ecosystems Won't Scale without Trust"**. 2017. URL: https://inform.tmforum.org/features-and-analysis/2017/01/digital-ecosystems-wont-wont-scale-without-trust/ (visited on 07/02/2021).

[GMR+21] G. GARIMELLA, P. MOHASSEL, M. ROSULEK, S. SADEGHIAN, J. SINGH. **"Private Set Operations from Oblivious Switching"**. In: *Public Key Cryptography (PKC)*. Vol. 12711. LNCS. Springer, 2021, pp. 591–617.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. **"How to Play any Mental Game or a Completeness Theorem for Protocols with Honest Majority"**. In: *ACM Symposium on Theory of Computing (STOC)*. ACM, 1987, pp. 218–229.

[Gon81] G. H. GONNET. **"Expected Length of the Longest Probe Sequence in Hash Code Searching"**. In: *Journal of the ACM (JACM)* 28.2 (1981). ACM, pp. 289–304.

[Gup16] S. GUPTA. **"Emerging Threats Abusing Phone Numbers Exploiting Cross-Platform Features"**. In: *Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE Computer Society, 2016, pp. 1339–1341.

[Ham21] I. A. HAMILTON. **"A Leaked Facebook PR Memo Shows How the Company Plans to Downplay Data Leaks Obtained through Scraping and Sidestep Criticism"**. 2021. URL: https://www.businessinsider.com/facebook-pr-memo-data-scraping-leaks-2021-4 (visited on 07/02/2021).

[HEK12] Y. HUANG, D. EVANS, J. KATZ. **"Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?"** In: *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2012.

[Hel80] M. E. HELLMAN. **"A Cryptanalytic Time-Memory Trade-Off"**. In: *IEEE Transactions on Information Theory* 26.4 (1980). IEEE Computer Society, pp. 401–406.

[HFH99] B. A. HUBERMAN, M. K. FRANKLIN, T. HOGG. **"Enhancing Privacy and Trust in Electronic Communities"**. In: *Electronic Commerce (EC)*. ACM, 1999, pp. 78–86.

[HHNZ19] M. HASTINGS, B. HEMENWAY, D. NOBLE, S. ZDANCEWIC. **"SoK: General Purpose Compilers for Secure Multi-Party Computation"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2019, pp. 1220–1237.

[HHS+21a] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop"**. In: *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. ACM, 2021, pp. 371–373.

[HHS+21b] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop"**. In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 3577–3594.

[HL10] C. HAZAY, Y. LINDELL. **"Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries"**. In: *Journal of Cryptology (JoC)* 23.3 (2010). Springer, pp. 422–456.

[HN10] C. HAZAY, K. NISSIM. **"Efficient Set Operations in the Presence of Malicious Adversaries"**. In: *Public Key Cryptography (PKC)*. Vol. 6056. LNCS. Springer, 2010, pp. 312–331.

[Hol21] A. HOLMES. **"533 Million Facebook Users' Phone Numbers and Personal Data Have Been Leaked Online"**. 2021. URL: https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4 (visited on 07/02/2021).

[HOS17] P. A. HALLGREN, C. ORLANDI, A. SABELFELD. **"PrivatePool: Privacy-Preserving Ridesharing"**. In: *Computer Security Foundations (CSF)*. IEEE Computer Society, 2017, pp. 276–291.

[HS20] A. HEINRICH, M. STUTE. **"OpenDrop: an Open Source AirDrop Implementation"**. 2020. URL: https://github.com/seemoo-lab/opendrop (visited on 07/02/2021).

[HST+21] T. HELDMANN, T. SCHNEIDER, O. TKACHENKO, C. WEINERT, H. YALAME. **"LLVM-based Circuit Compilation for Practical Secure Computation"**. In: *Applied Cryptography and Network Security (ACNS)*. Vol. 12727. LNCS. Springer, 2021, pp. 99–121.

[Hun19] T. HUNT. **"The 773 Million Record Collection #1 Data Breach"**. 2019. URL: https://www.troyhunt.com/the-773-million-record-collection-1-data-reach/ (visited on 07/02/2021).

[HWS+21] C. HAGEN, C. WEINERT, C. SENDNER, A. DMITRIENKO, T. SCHNEIDER. **"All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers"**. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2021.

[IKN+17] M. ION, B. KREUTER, E. NERGIZ, S. PATEL, S. SAXENA, K. SETH, D. SHANAHAN, M. YUNG. **"Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions"**. In: *Cryptology ePrint Archive* 2017/738 (2017). IACR.

[IKN+20] M. ION, B. KREUTER, A. E. NERGIZ, S. PATEL, S. SAXENA, K. SETH, M. RAYKOVA, D. SHANAHAN, M. YUNG. **"On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality"**. In: *IEEE European Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2020, pp. 370–389.

[IKNP03] Y. ISHAI, J. KILIAN, K. NISSIM, E. PETRANK. **"Extending Oblivious Transfers Efficiently"**. In: *Advances in Cryptology – CRYPTO*. Vol. 2729. LNCS. Springer, 2003, pp. 145–161.

[Int19] INTEL CORPORATION. **"Intel Software Guard Extensions (Intel SGX)"**. 2019. URL: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html (visited on 07/02/2021).

[JL09a] S. JARECKI, X. LIU. **"Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection"**. In: *Theory of Cryptography Conference (TCC)*. Vol. 5444. LNCS. Springer, 2009, pp. 577–594.

[JL09b] S. JARECKI, X. LIU. **"Private Mutual Authentication and Conditional Oblivious Transfer"**. In: *Advances in Cryptology – CRYPTO*. Vol. 5677. LNCS. Springer, 2009, pp. 90–107.

[JL10] S. JARECKI, X. LIU. **"Fast Secure Computation of Set Intersection"**. In: *Security and Cryptography for Networks (SCN)*. Vol. 6280. LNCS. Springer, 2010, pp. 418–435.

[JNS17] N. M. JOHNSON, J. P. NEAR, D. X. SONG. **"Practical Differential Privacy for SQL Queries Using Elastic Sensitivity"**. In: *Computing Research Repository (CoRR)* abs/1706.09479 (2017). arXiv preprint.

[JNS18] N. M. JOHNSON, J. P. NEAR, D. SONG. **"Towards Practical Differential Privacy for SQL Queries"**. In: *Proceedings of the VLDB Endowment* 11.5 (2018). VLDB Endowment, pp. 526–539.

[Kas21] J. KASTRENAKES. **"Apple Says There Are Now Over 1 Billion Active iPhones"**. 2021. URL: https://www.theverge.com/2021/1/27/22253162/iphone-users-total-number-billion-apple-tim-cook-q1-2021 (visited on 07/02/2021).

[KK20] F. KARAKOÇ, A. KÜPÇÜ. **"Linear Complexity Private Set Intersection for Secure Two-Party Protocols"**. In: *Cryptology And Network Security (CANS)*. Vol. 12579. LNCS. Springer, 2020, pp. 409–429.

[KKC+17] J. KIM, K. KIM, J. CHO, H. KIM, S. SCHRITTWIESER. **"Hello, Facebook! Here Is the Stalkers' Paradise!: Design and Analysis of Enumeration Attack Using Phone Numbers on Facebook"**. In: *Information Security Practice and Experience (ISPEC)*. Vol. 10701. LNCS. Springer, 2017, pp. 663–677.

[KKL+20] B. KACSMAR, B. KHURRAM, N. LUKAS, A. NORTON, M. SHAFIEINEJAD, Z. SHANG, Y. BASERI, M. SEPEHRI, S. OYA, F. KERSCHBAUM. **"Differentially Private Two-Party Set Operations"**. In: *IEEE European Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2020, pp. 390–404.

[KKRT16] V. KOLESNIKOV, R. KUMARESAN, M. ROSULEK, N. TRIEU. **"Efficient Batched Oblivious PRF with Applications to Private Set Intersection"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 818–829.

[Klo17] L. KLOEZE. **"Collecting Huge Amounts of Data with WhatsApp"**. 2017. URL: https://www.lorankloeze.nl/2017/05/07/collecting-huge-amounts-of-data-with-whatsapp/ (visited on 07/02/2021).

[KLS+17] Á. KISS, J. LIU, T. SCHNEIDER, N. ASOKAN, B. PINKAS. **"Private Set Intersection for Unequal Set Sizes with Mobile Applications"**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.4 (2017). De Gruyter Open, pp. 177–197.

[KMP+17] V. KOLESNIKOV, N. MATANIA, B. PINKAS, M. ROSULEK, N. TRIEU. **"Practical Multi-party Private Set Intersection from Symmetric-Key Techniques"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 1257–1272.

[KMW08] A. KIRSCH, M. MITZENMACHER, U. WIEDER. **"More Robust Hashing: Cuckoo Hashing with a Stash"**. In: *European Symposium on Algorithms (ESA)*. Vol. 5193. LNCS. Springer, 2008, pp. 611–622.

[KNR+17] S. KRÜGER, S. NADI, M. REIF, K. ALI, M. MEZINI, E. BODDEN, F. GÖPFERT, F. GÜNTHER, C. WEINERT, D. DEMMLER, R. KAMATH. **"CogniCrypt: Supporting Developers in using Cryptography"**. In: *Automated Software Engineering (ASE)*. IEEE Computer Society, 2017, pp. 931–936.

[KOS15] M. KELLER, E. ORSINI, P. SCHOLL. **"Actively Secure OT Extension with Optimal Overhead"**. In: *Advances in Cryptology – CRYPTO*. Vol. 9215. LNCS. Springer, 2015, pp. 724–741.

[KPKS14] E. KIM, K. PARK, H. KIM, J. SONG. **"I've Got Your Number: - Harvesting Users' Personal Data via Contacts Sync for the KakaoTalk Messenger"**. In: *Workshop on Information Security Applications (WISA)*. Vol. 8909. LNCS. Springer, 2014, pp. 55–67.

[KPKS15] E. KIM, K. PARK, H. KIM, J. SONG. **"Design and Analysis of Enumeration Attacks on Finding Friends with Phone Numbers: A Case Study with KakaoTalk"**. In: *Computers & Security* 52 (2015). Elsevier, pp. 267–275.

[KRS+19] D. KALES, C. RECHBERGER, T. SCHNEIDER, M. SENKER, C. WEINERT. **"Mobile Private Contact Discovery at Scale"**. In: *USENIX Security Symposium*. USENIX Association, 2019, pp. 1447–1464.

[KS05] L. KISSNER, D. X. SONG. **"Privacy-Preserving Set Operations"**. In: *Advances in Cryptology – CRYPTO*. Vol. 3621. LNCS. Springer, 2005, pp. 241–257.

[KS08] V. KOLESNIKOV, T. SCHNEIDER. **"Improved Garbled Circuit: Free XOR Gates and Applications"**. In: *International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 5126. LNCS. Springer, 2008, pp. 486–498.

[KZ20] D. KALES, G. ZAVERUCHA. **"Improving the Performance of the Picnic Signature Scheme"**. In: *Cryptology ePrint Archive* 2020/427 (2020). IACR.

[LCWZ14] D. LAZAR, H. CHEN, X. WANG, N. ZELDOVICH. **"Why Does Cryptographic Software Fail?: A Case Study and Open Problems"**. In: *Asia-Pacific Workshop on Systems (APSys)*. ACM, 2014, 7:1–7:7.

[LIM21a] F. LIU, T. ISOBE, W. MEIER. **"A Simple Algebraic Attack on 3-Round LowMC"**. In: *Cryptology ePrint Archive* 2021/255 (2021). IACR.

[LIM21b] F. LIU, T. ISOBE, W. MEIER. **"Cryptanalysis of Full LowMC and LowMC-M with Algebraic Techniques"**. In: *Advances in Cryptology – CRYPTO*. Vol. 12827. LNCS. Springer, 2021, pp. 368–401.

[LPR+20] T. LEPOINT, S. PATEL, M. RAYKOVA, K. SETH, N. TRIEU. **"Private Join and Compute from PIR with Default"**. In: *Cryptology ePrint Archive* 2020/1011 (2020). IACR.

[LWN+15] C. LIU, X. S. WANG, K. NAYAK, Y. HUANG, E. SHI. **"ObliVM: A Programming Framework for Secure Computation"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2015, pp. 359–376.

[Mar08] S. MARECHAL. **"Advances in Password Cracking"**. In: *Journal of Computer Virology and Hacking Techniques* 4.1 (2008). Springer, pp. 73–81.

[Mar14] M. MARLINSPIKE. **"The Difficulty of Private Contact Discovery"**. 2014. URL: https://signal.org/blog/contact-discovery/ (visited on 07/02/2021).

[Mar17] M. MARLINSPIKE. **"Technology Preview: Private Contact Discovery for Signal"**. 2017. URL: https://signal.org/blog/private-contact-discovery/ (visited on 07/02/2021).

[Mea86] C. A. MEADOWS. **"A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party"**. In: *IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 1986, pp. 134–137.

[MNPS04] D. MALKHI, N. NISAN, B. PINKAS, Y. SELLA. **"Fairplay – Secure Two-Party Computation System"**. In: *USENIX Security Symposium*. USENIX Association, 2004, pp. 287–302.

[MPR+20] P. MIAO, S. PATEL, M. RAYKOVA, K. SETH, M. YUNG. **"Two-Sided Malicious Security for Private Intersection-Sum with Cardinality"**. In: *Advances in Cryptology – CRYPTO*. Vol. 12172. LNCS. Springer, 2020, pp. 3–33.

[MRR20] P. MOHASSEL, P. RINDAL, M. ROSULEK. **"Fast Database Joins and PSI for Secret Shared Data"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2020, pp. 1271–1287.

[MS13] P. MOHASSEL, S. S. SADEGHIAN. **"How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 7881. LNCS. Springer, 2013, pp. 557–574.

[MSF+14] R. MUELLER, S. SCHRITTWIESER, P. FRÜHWIRT, P. KIESEBERG, E. R. WEIPPL. **"What's new with WhatsApp & Co.? Revisiting the Security of Smartphone Messaging Applications"**. In: *Information Integration and Web-based Applications & Services (iiWAS)*. ACM, 2014, pp. 142–151.

[MZM+18] M. MARX, E. ZIMMER, T. MUELLER, M. BLOCHBERGER, H. FEDERRATH. **"Hashing of Personally Identifiable Information Is Not Sufficient"**. In: *Sicherheit*. Vol. P-281. LNI. Gesellschaft für Informatik e.V., 2018, pp. 55–68.

[New21] L. H. NEWMAN. **"Facebook Had Years to Fix the Flaw That Leaked 500M Users' Data"**. 2021. URL: https://www.wired.com/story/facebook-data-leak-contact-import-flaws/ (visited on 07/02/2021).

[NH12] A. NARAYAN, A. HAEBERLEN. **"DJoin: Differentially Private Join Queries over Distributed Databases"**. In: *Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2012, pp. 149–162.

[NR04] M. NAOR, O. REINGOLD. **"Number-Theoretic Constructions of Efficient Pseudo-Random Functions"**. In: *Journal of the ACM (JACM)* 51.2 (2004). ACM, pp. 231–262.

[OCS+21] J. ONDRUSEK, D. COULTER, L. SALDANHA, L. KELLER, B. HERMANSEN, G. LINDSAY, A. M. GORZELANY, L. POGGEMEYER, D. HALFIN, J. DECKER, E. GALLAGHER, A. KIM. **"Optimize Windows 10 Update Delivery"**. 2021. URL: https://docs.microsoft.com/en-us/windows/deployment/update/waas-optimize-windows-10-updates (visited on 07/02/2021).

[Oec03] P. OECHSLIN. **"Making a Faster Cryptanalytic Time-Memory Trade-Off"**. In: *Advances in Cryptology – CRYPTO*. Vol. 2729. LNCS. Springer, 2003, pp. 617–630.

[OLe18] R.-R. O'LEARY. **"Ethereum ASICs Are Here"**. 2018. URL: https://www.coindesk.com/ethereum-asics-means-whats-next (visited on 07/02/2021).

[Pai99] P. PAILLIER. **"Public-Key Cryptosystems Based on Composite Degree Residuosity Classes"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 1592. LNCS. Springer, 1999, pp. 223–238.

[PKY+21] R. PODDAR, S. KALRA, A. YANAI, R. DENG, R. A. POPA, J. M. HELLERSTEIN. **"Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics"**. In: *USENIX Security Symposium*. USENIX Association, 2021.

[PNH17] A. PAPADIMITRIOU, A. NARAYAN, A. HAEBERLEN. **"DStress: Efficient Differentially Private Computations on Distributed Data"**. In: *European Conference on Computer Systems (EuroSys)*. ACM, 2017, pp. 560–574.

[PR01] R. PAGH, F. F. RODLER. **"Cuckoo Hashing"**. In: *European Symposium on Algorithms (ESA)*. Vol. 2161. LNCS. Springer, 2001, pp. 121–133.

[Pri21] R. PRICE. **"Researchers Warned for Years That Facebook's Phone Number Lookup Tools Were Ripe for Abuse"**. 2021. URL: https://www.businessinsider.com/facebook-hack-phone-number-whatsapp-vulnerability-user-data-researchers-2021-4 (visited on 07/02/2021).

[PRTY19] B. PINKAS, M. ROSULEK, N. TRIEU, A. YANAI. **"SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension"**. In: *Advances in Cryptology – CRYPTO*. Vol. 11694. LNCS. Springer, 2019, pp. 401–431.

[PRTY20] B. PINKAS, M. ROSULEK, N. TRIEU, A. YANAI. **"PSI from PaXoS: Fast, Malicious Private Set Intersection"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 12106. LNCS. Springer, 2020, pp. 739–767.

[PSSW09] B. PINKAS, T. SCHNEIDER, N. P. SMART, S. C. WILLIAMS. **"Secure Two-Party Computation Is Practical"**. In: *Advances in Cryptology – ASIACRYPT*. Vol. 5912. LNCS. Springer, 2009, pp. 250–267.

[PSSZ15] B. PINKAS, T. SCHNEIDER, G. SEGEV, M. ZOHNER. **"Phasing: Private Set Intersection Using Permutation-based Hashing"**. In: *USENIX Security Symposium*. USENIX Association, 2015, pp. 515–530.

[PSTY19] B. PINKAS, T. SCHNEIDER, O. TKACHENKO, A. YANAI. **"Efficient Circuit-Based PSI with Linear Communication"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 11478. LNCS. Springer, 2019, pp. 122–153.

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 10822. LNCS. Springer, 2018, pp. 125–157.

[PSZ14] B. PINKAS, T. SCHNEIDER, M. ZOHNER. **"Faster Private Set Intersection Based on OT Extension"**. In: *USENIX Security Symposium*. USENIX Association, 2014, pp. 797–812.

[PSZ18] B. PINKAS, T. SCHNEIDER, M. ZOHNER. **"Scalable Private Set Intersection Based on OT Extension"**. In: *ACM Transactions on Privacy and Security (TOPS)* 21.2 (2018). ACM, 7:1–7:35.

[RA18] A. C. D. RESENDE, D. F. ARANHA. **"Faster Unbalanced Private Set Intersection"**. In: *Financial Cryptography and Data Security (FC)*. Vol. 10957. LNCS. Springer, 2018, pp. 203–221.

[RR17a] P. RINDAL, M. ROSULEK. **"Improved Private Set Intersection Against Malicious Adversaries"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 10210. LNCS. 2017, pp. 235–259.

[RR17b] P. RINDAL, M. ROSULEK. **"Malicious-Secure Private Set Intersection via Dual Execution"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 1229–1242.

[RR21] M. ROSULEK, L. ROY. **"Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits"**. In: *Advances in Cryptology – CRYPTO*. Vol. 12825. LNCS. Springer, 2021, pp. 94–124.

[RS21] P. RINDAL, P. SCHOPPMANN. **"VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 12697. LNCS. Springer, 2021, pp. 901–930.

[RVC16] Y. RASHIDI, K. VANIEA, L. J. CAMP. **"Understanding Saudis' Privacy Concerns When Using WhatsApp"**. In: *Workshop on Usable Security (USEC)*. Internet Society, 2016.

[RWT+18] M. S. RIAZI, C. WEINERT, O. TKACHENKO, E. M. SONGHORI, T. SCHNEIDER, F. KOUSHANFAR. **"Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications"**. In: *ACM Asia Conference on Computer and Communication Security (AsiaCCS)*. ACM, 2018, pp. 707–721.

[SC20] M. SHILLING, A. CELNER. **"2021 Banking and Capital Markets Outlook"**. 2020. URL: https://www2.deloitte.com/global/en/insights/industry/financial-services/financial-services-industry-outlooks/banking-industry-outlook.html (visited on 07/02/2021).

[Sch20] D. M. SCHWAYCER. **"Instantly Share Files with People around You with Nearby Share"**. 2020. URL: https://blog.google/products/android/nearby-share/ (visited on 07/02/2021).

[SFK+12] S. SCHRITTWIESER, P. FRÜHWIRT, P. KIESEBERG, M. LEITHNER, M. MULAZZANI, M. HUBER, E. R. WEIPPL. **"Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications"**. In: *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2012.

[SGRR19] P. SCHOPPMANN, A. GASCÓN, L. REICHERT, M. RAYKOVA. **"Distributed Vector-OLE: Improved Constructions and Implementation"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 1055–1072.

[Sha80] A. SHAMIR. **"On the Power of Commutativity in Cryptography"**. In: *International Colloquium on Automata, Languages and Programming (ICALP)*. Vol. 85. LNCS. Springer, 1980, pp. 582–595.

[Sha84] A. SHAMIR. **"Identity-Based Cryptosystems and Signature Schemes"**. In: *Advances in Cryptology – CRYPTO*. Vol. 196. LNCS. Springer, 1984, pp. 47–53.

[Sig20] R. SIGNAL. **"Introducing Signal PINs"**. 2020. URL: https://signal.org/blog/signal-pins/ (visited on 07/02/2021).

[SKH18a] M. STUTE, D. KREITSCHMANN, M. HOLLICK. **"One Billion Apples' Secret Sauce: Recipe for the *Apple Wireless Direct Link* Ad hoc Protocol"**. In: *Mobile Computing and Networking (MobiCom)*. ACM, 2018, pp. 529–543.

[SKH18b] M. STUTE, D. KREITSCHMANN, M. HOLLICK. **"The Open Wireless Link Project"**. 2018. URL: https://owlink.org (visited on 07/02/2021).

[SNM+19] M. STUTE, S. NARAIN, A. MARIOTTO, A. HEINRICH, D. KREITSCHMANN, G. NOUBIR, M. HOLLICK. **"A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link"**. In: *USENIX Security Symposium*. USENIX Association, 2019, pp. 37–54.

[Sto21] L. STOCKLEY. **"How a WhatsApp Status Loophole Is Aiding Cyberstalkers"**. 2021. URL: https://traced.app/2021/04/13/whatsapp-status-loophole-is-aiding-cyberstalkers/ (visited on 07/02/2021).

[Tat15] E. I. TATLI. **"Cracking More Password Hashes With Patterns"**. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 10.8 (2015). IEEE Computer Society, pp. 1656–1665.

[TPY+19] K. THOMAS, J. PULLMAN, K. YEO, A. RAGHUNATHAN, P. G. KELLEY, L. INVERNIZZI, B. BENKO, T. PIETRASZEK, S. PATEL, D. BONEH, E. BURSZTEIN. **"Protecting Accounts from Credential Stuffing with Password Breach Alerting"**. In: *USENIX Security Symposium*. USENIX Association, 2019, pp. 1556–1571.

[TWSH18] O. TKACHENKO, C. WEINERT, T. SCHNEIDER, K. HAMACHER. **"Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies"**. In: *ACM Asia Conference on Computer and Communication Security (AsiaCCS)*. ACM, 2018, pp. 221–235.

[VC05] J. VAIDYA, C. CLIFTON. **"Secure Set Intersection Cardinality with Application to Association Rule Mining"**. In: *Journal of Computer Security* 13.4 (2005). IOS Press, pp. 593–622.

[VSG+19] N. VOLGUSHEV, M. SCHWARZKOPF, B. GETCHELL, M. VARIA, A. LAPETS, A. BESTAVROS. **"Conclave: Secure Multi-Party Computation on Big Data"**. In: *European Conference on Computer Systems (EuroSys)*. ACM, 2019, 3:1–3:18.

[Wak68] A. WAKSMAN. **"A Permutation Network"**. In: *Journal of the ACM (JACM)* 15.1 (1968). ACM, pp. 159–163.

[WE21] J. C. WONG, J. ERNST. **"Facebook Knew of Honduran President's Manipulation Campaign"**. 2021. URL: https://www.theguardian.com/technology/2021/apr/13/facebook-honduras-juan-orlando-hernandez-fake-engagement (visited on 07/02/2021).

[Wie17] U. WIEDER. **"Hashing, Load Balancing and Multiple Choice"**. In: *Foundations and Trends in Theoretical Computer Science* 12.3-4 (2017). now publishers, pp. 275–379.

[WTSB16] D. J. WU, A. TALY, A. SHANKAR, D. BONEH. **"Privacy, Discovery, and Authentication for the Internet of Things"**. In: *European Symposium on Research in Computer Security (ESORICS)*. Vol. 9879. LNCS. Springer, 2016, pp. 301–319.

[WYKW20] C. WENG, K. YANG, J. KATZ, X. WANG. **"Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits"**. In: *Cryptology ePrint Archive* 2020/925 (2020). IACR.

[Yao82] A. C.-C. YAO. **"Protocols for Secure Computations (Extended Abstract)"**. In: *Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1982, pp. 160–164.

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets (Extended Abstract)"**. In: *Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1986, pp. 162–167.

[YBAG04] J. J. YAN, A. F. BLACKWELL, R. J. ANDERSON, A. GRANT. **"Password Memorability and Security: Empirical Results"**. In: *IEEE Security and Privacy (S&P)* 2.5 (2004). IEEE Computer Society, pp. 25–31.

[YCP+20] J. H. M. YING, S. CAO, G. S. POH, J. XU, H. W. LIM. **"PSI-Stats: Private Set Intersection Protocols Supporting Secure Statistical Functions"**. In: *Cryptology ePrint Archive* 2020/623 (2020). IACR.

[YWL+20] K. YANG, C. WENG, X. LAN, J. ZHANG, X. WANG. **"Ferret: Fast Extension for Correlated OT with Small Communication"**. In: *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2020, pp. 1607–1626.

[ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **"Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates"**. In: *Advances in Cryptology – EUROCRYPT*. Vol. 9057. LNCS. Springer, 2015, pp. 220–250.

[Zwe15a] M. ZWEERINK. **"WhatsApp Privacy is Broken!"** 2015. URL: https://maikel.pro/blog/en-whatsapp-privacy-options-are-illusions/ (visited on 07/02/2021).

[Zwe15b] M. ZWEERINK. **"WhatsApp Privacy Problem Explained in Detail"**. 2015. URL: https://maikel.pro/blog/en-whatsapp-privacy-problem-explained-in-detail/ (visited on 07/02/2021).

[Zwe16] M. ZWEERINK. **"PoC WhatsSpy Public Support Ending Today"**. 2016. URL: https://maikel.pro/blog/whatsspy-public-support-ending-today (visited on 07/02/2021).

List of Figures

List of Tables

List of Abbreviations

ASIC	Application-Specific Integrated Circuit
AWDL	Apple Wireless Direct Link
BLE	Bluetooth Low Energy
CA	Certification Authority
CE	Cryptography Extensions
CRS	Common Reference String
DDH	Decisional Diffie-Hellman
EFF	Electronic Frontier Foundation
FHE	Fully Homomorphic Encryption
FPGA	Field-Programmable Gate Array
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HE	Homomorphic Encryption
IBE	Identity-based Encryption
ITU	International Telecommunication Union
JNI	Java Native Interface
MPC	Multi-Party Computation
ORAM	Oblivious Random Access Machine
OPRF	Oblivious Pseudo-Random Function
OPPRF	Oblivious Programmable Pseudo-Random Function
OT	Oblivious Transfer
PII	Personally Identifiable Information
PIR	Private Information Retrieval
PKI	Public Key Infrastructure
PSI	Private Set Intersection
PSM	Private Set Membership
q**-DHI**	q-Diffie-Hellman Inversion
ROM	Random Oracle Model
RTT	Round Trip Time
SCS	Sort-Compare-Shuffle
SGX	Software Guard Extensions
TEE	Trusted Execution Environment
UUID	Universally Unique Identifier
VOLE	Vector Oblivious Linear Evaluation

List of Own Publications

Peer-reviewed Publications

[BFJ+20] S. P. Bayerl, T. Frassetto, P. Jauernig, K. Riedhammer, A.-R. Sadeghi, T. Schneider, E. Stapf, C. Weinert. **"Offline Model Guard: Secure and Private ML on Mobile Devices"**. In: *23. Design, Automation & Test in Europe Conference & Exhibition (DATE'20)*. IEEE Computer Society, 2020, pp. 460–465. CORE Rank B.

[BFR+18] F. Brasser, T. Frassetto, K. Riedhammer, A.-R. Sadeghi, T. Schneider, C. Weinert. **"VoiceGuard: Secure and Private Speech Processing"**. In: *19. Conference of the International Speech Communication Association (INTERSPEECH'18)*. International Speech Communication Association (ISCA), 2018, pp. 1303–1307. CORE Rank A.

[DKS+21] D. Demmler, S. Katzenbeisser, T. Schneider, T. Schuster, C. Weinert. **"Improved Circuit Compilation for Hybrid MPC via Compiler Intermediate Representation"**. In: *18. International Conference on Security and Cryptography (SECRYPT'21)*. Short paper. SciTePress, 2021, pp. 444–451. CORE Rank B.

[FKSW19] S. Felsen, Á. Kiss, T. Schneider, C. Weinert. **"Secure and Private Function Evaluation with Intel SGX"**. In: *10. ACM Cloud Computing Security Workshop (CCSW'19)*. ACM, 2019, pp. 165–181.

[HWS+21] C. Hagen, C. Weinert, C. Sendner, A. Dmitrienko, T. Schneider. **"All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers"**. In: *28. Network and Distributed System Security Symposium (NDSS'21)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2020/1119. Internet Society, 2021. CORE Rank A*. Appendix A.

[HHS+21a] A. Heinrich, M. Hollick, T. Schneider, M. Stute, C. Weinert. **"DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop"**. In: *14. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/893. ACM, 2021, pp. 371–373. Appendix C.

[HHS+21b] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop"**. In: *30. USENIX Security Symposium (USENIX Security'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/481. USENIX Association, 2021, pp. 3577–3594. CORE Rank A*. Appendix D.

[HST+21] T. HELDMANN, T. SCHNEIDER, O. TKACHENKO, C. WEINERT, H. YALAME. **"LLVM-based Circuit Compilation for Practical Secure Computation"**. In: *19. International Conference on Applied Cryptography and Network Security (ACNS'21)*. Vol. 12727. LNCS. Springer, 2021, pp. 99–121. CORE Rank B.

[KRS+19] D. KALES, C. RECHBERGER, T. SCHNEIDER, M. SENKER, C. WEINERT. **"Mobile Private Contact Discovery at Scale"**. In: *28. USENIX Security Symposium (USENIX Security'19)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2019/517. USENIX Association, 2019, pp. 1447–1464. CORE Rank A*. Appendix B.

[KNR+17] S. KRÜGER, S. NADI, M. REIF, K. ALI, M. MEZINI, E. BODDEN, F. GÖPFERT, F. GÜNTHER, C. WEINERT, D. DEMMLER, R. KAMATH. **"CogniCrypt: Supporting Developers in using Cryptography"**. In: *32. IEEE/ACM International Conference on Automated Software Engineering (ASE'17)*. IEEE Computer Society, 2017, pp. 931–936. CORE Rank A*.

[MSS+20] H. MANTEL, L. SCHEIDEL, T. SCHNEIDER, A. WEBER, C. WEINERT, T. WEISSMANTEL. **"RiCaSi: Rigorous Cache Side Channel Mitigation via Selective Circuit Compilation"**. In: *19. International Conference on Cryptology And Network Security (CANS'20)*. Vol. 12579. LNCS. Springer, 2020, pp. 505–525. CORE Rank B.

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *37. Advances in Cryptology – EUROCRYPT'18*. Vol. 10822. LNCS. Code: https://encrypto.de/code/2DCH. Full version: https://ia.cr/2018/120. Springer, 2018, pp. 125–157. CORE Rank A*. Appendix E.

[RWT+18] M. S. RIAZI, C. WEINERT, O. TKACHENKO, E. M. SONGHORI, T. SCHNEIDER, F. KOUSHANFAR. **"Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications"**. In: *13. ACM Asia Conference on Computer and Communications Security (AsiaCCS'18)*. ACM, 2018, pp. 707–721. CORE Rank A.

[TWSH18] O. TKACHENKO, C. WEINERT, T. SCHNEIDER, K. HAMACHER. **"Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies"**. In: *13. ACM Asia Conference on Computer and Communications Security (AsiaCCS'18)*. ACM, 2018, pp. 221–235. CORE Rank A.

[TMW+20a] A. TREIBER, A. MOLINA, C. WEINERT, T. SCHNEIDER, K. KERSTING. **"CryptoSPN: Expanding PPML beyond Neural Networks"**. In: *Privacy-Preserving Machine Learning in Practice (PPMLP@CCS'20) – CCS'20 Workshop*. ACM, 2020, pp. 9–14.

[TMW+20b] A. TREIBER, A. MOLINA, C. WEINERT, T. SCHNEIDER, K. KERSTING. **"CryptoSPN: Privacy-preserving Sum-Product Network Inference"**. In: *24. European Conference on Artificial Intelligence (ECAI'20)*. IOS Press, 2020, pp. 1946–1953. CORE Rank A.

[VBC+15] M. A. G. VIGIL, J. BUCHMANN, D. CABARCAS, C. WEINERT, A. WIESMAIER. **"Integrity, Authenticity, Non-Repudiation, and Proof of Existence for Long-Term Archiving: A Survey"**. In: *Computers & Security* 50 (2015), pp. 16–32. CORE Rank B.

[VWB+14] M. A. G. VIGIL, C. WEINERT, K. BRADEN, D. DEMIREL, J. BUCHMANN. **"A Performance Analysis of Long-Term Archiving Techniques"**. In: *16. International Conference on High Performance Computing and Communications (HPCC'14)*. IEEE Computer Society, 2014, pp. 878–889. CORE Rank B.

[VWDB14] M. A. G. VIGIL, C. WEINERT, D. DEMIREL, J. BUCHMANN. **"An Efficient Time-Stamping Solution for Long-Term Digital Archiving"**. In: *33. International Performance Computing and Communications Conference (IPCCC'14)*. IEEE Computer Society, 2014, pp. 1–8. CORE Rank B.

[WDV+17] C. WEINERT, D. DEMIREL, M. A. G. VIGIL, M. GEIHS, J. BUCHMANN. **"MoPS: A Modular Protection Scheme for Long-Term Storage"**. In: *12. ACM Asia Conference on Computer and Communications Security (AsiaCCS'17)*. ACM, 2017, pp. 436–448. CORE Rank A.

Technical Reports

[CSR+20] R. CAMMAROTA, M. SCHUNTER, A. RAJAN, F. BOEMER, Á. KISS, A. TREIBER, C. WEINERT, T. SCHNEIDER, E. STAPF, A.-R. SADEGHI, D. DEMMLER, H. CHEN, S. U. HUSSAIN, S. RIAZI, F. KOUSHANFAR, S. GUPTA, T. S. ROSING, K. CHAUDHURI, H. NEJATOLLAHI, N. DUTT, M. IMANI, K. LAINE, A. DUBEY, A. AYSU, F. S. HOSSEINI, C. YANG, E. WALLACE, P. NORTON. **"Trustworthy AI Inference Systems: An Industry Research View"**. https://arxiv.org/abs/2008.04449. 2020. arXiv: 2008.04449 [cs.CR].

Christian Weinert

Short CV

Date of Birth 6th September 1991

E-Mail weinert@encrypto.cs.tu-darmstadt.de

Education

09/2016 - 08/2021 **Ph.D. Student**, TU Darmstadt
- Advisor: Prof. Dr.-Ing. Thomas Schneider
- Thesis: *Practical Private Set Intersection Protocols for Privacy-Preserving Applications*

10/2013 - 07/2016 **M.Sc. Computer Science**, TU Darmstadt, *GPA: with honors* (1.06)
- Minor: Business Administration, Economics and Law
- Thesis: *Building a Modular Long-Term Archiving System*

10/2010 - 09/2013 **B.Sc. Computer Science**, TU Darmstadt, *GPA: very good* (1.25)
- Thesis: *Content Integrity System and Notarisation for Long-Term Protection of Data Objects*

Work and Research Experience

TU Darmstadt

09/2016 - 08/2021 **Doctoral Researcher**, *Cryptography and Privacy Engineering Group* (ENCRYPTO, headed by Prof. Dr.-Ing. Thomas Schneider), TU Darmstadt

03/2016 - 07/2016
04/2014 - 03/2015 **Student Research Assistant**, *Cryptography and Computer Algebra* (CDC, headed by Prof. Dr. Johannes Buchmann), TU Darmstadt

Project Affiliations

since 01/2021 **Engineering Private AI Systems** (EPAI) at the *Private AI Collaborative Research Institute*, funded by Intel, Avast, and Borsetta

since 01/2019 **Practical Private Set Intersection for Data Protection** (as part of the mission "Future Data Economy and Society: Co-Development of Privacy-Enhancing Technologies and Data-Driven Business Models") at the *National Research Center for Applied Cybersecurity* (ATHENE), funded by the *German Federal Ministry of Education and Research* (BMBF) and the *Federal State of Hesse*

since 09/2016 **Compiler for Privacy-Preserving Protocols** (E4) at the *Collaborative Research Center* (CRC) *Cryptography-Based Security Solutions* (CROSSING), funded by the *German Research Foundation* (DFG)

06/2017 - 12/2019 **Protection Mechanisms for Big Data and Complex Private Functions** (IP3) at the *Center for Research in Security and Privacy* (CRISP), funded by the *German Federal Ministry of Education and Research* (BMBF) and the *Federal State of Hesse*

09/2016 - 03/2018 **Scalable Privacy-Preserving Protocols** (IP1) at the *Center for Research in Security and Privacy* (CRISP), funded by the *German Federal Ministry of Education and Research* (BMBF) and the *Federal State of Hesse*

03/2016 - 07/2016
04/2014 - 03/2015 **Long-Term Secure Archiving** (S6) at the *Collaborative Research Center* (CRC) *Cryptography-Based Security Solutions* (CROSSING), funded by the *German Research Foundation* (DFG)

Teaching

Winter Term 20/21	**Lecturer** (stand-in for 4 lectures) and **Teaching Assistant**, *Digitaltechnik*, TU Darmstadt (>950 students)
Summer Term 20	**Teaching Assistant**, *Cryptographic Protocols*, TU Darmstadt **Seminar**, *Privacy-Preserving Technologies* (PrivTech), TU Darmstadt
Winter Term 19/20	**Teaching Assistant**, *Digitaltechnik*, TU Darmstadt (>850 students)
Summer Term 19	**Teaching Assistant**, *Cryptographic Protocols* (CRYPROT), TU Darmstadt **Project/Lab**, *Development for Protecting Privacy* (PrivDev), TU Darmstadt
Winter Term 18/19	**Seminar**, *Privacy-Preserving Technologies* (PrivTech), TU Darmstadt **Project/Lab**, *Development for Protecting Privacy* (PrivDev), TU Darmstadt
Summer Term 18	**Teaching Assistant**, *Cryptographic Protocols* (CRYPROT), TU Darmstadt
Winter Term 17/18	**Seminar**, *Privacy-Preserving Technologies* (PrivTech), TU Darmstadt
Winter Term 16/17	**Seminar**, *Privacy-Preserving Technologies* (PrivTech), TU Darmstadt
Winter Term 15/16	**Student Teaching Assistant**, *Information Visualization and Visual Analytics* at the department of *Interactive Graphics Systems* (GRIS), TU Darmstadt and at the department of *Information Visualization and Visual Analytics*, Fraunhofer IGD

Supervised Master Students

09/2019 - 5/2020	**Nanako Honda**, *Privacy-Preserving Authentication Protocol For Apple AirDrop*, co-supervised with Prof. Dr.-Ing. Matthias Hollick and Dr.-Ing. Milan Stute
07/2018 - 01/2019	**Susanne Felsen**, *Secure Two-Party Computation: ABY versus Intel SGX*, co-supervised with Prof. Dr.-Ing. Thomas Schneider
12/2017 - 07/2018	**Matthias Senker**, *PSI meets Signal: Integrating a Malicious-Secure Private Contact Discover Solution in an Open-Source Instant Messaging Service*, co-supervised with Prof. Dr.-Ing. Thomas Schneider, award for "Best Master Thesis 2018" by "Friends of TU Darmstadt e.V."
04/2017 - 09/2017	**Oleksandr Tkachenko**, *Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies*, co-supervised with Prof. Dr.-Ing. Thomas Schneider and Prof. Dr. Kay Hamacher

Supervised Bachelor Students

02/2021 - 09/2021	**Roman Hergenreder**, *Privacy-Preserving Household Finance Analytics*, co-supervised with Prof. Dr.-Ing. Thomas Schneider
02/2020 - 09/2020	**Tim Heldmann**, *LLVM-based Circuit Compilation for Pratical Secure Computation*, co-supervised with M.Sc. Oleksandr Tkachenko and Prof. Dr.-Ing. Thomas Schneider
05/2019 - 11/2019	**Tom Schuster**, *Optimizing HyCC with Compiler Intermediate Representation*, co-supervised with Prof. Dr.-Ing. Thomas Schneider

Supervised Research Interns

04/2020 - 05/2020	**Deepak Kumaraswamy**, *Generic Multi-Party Private Set Intersection*, co-supervised with Prof. Dr.-Ing. Thomas Schneider

Supervised Student Research Assistants

since 08/2018	**Oliver Schick**
05/2019 - 04/2020	**Lukas Scheidel**
06/2018 - 01/2019	**Lennart Braun**
01/2017 - 09/2017	**Oleksandr Tkachenko**
04/2017 - 06/2017	**Prankur Chauhan**

Professional Training

04/2021 **Conflict Management**, Susanne Lörx
11/2020 **Slidewriting**, Dr. Markus Burger (Slidewriting.com)
11/2020 **Vision Training at the Computer**, Brigitte Göth (eye-eXercise)
10/2019 **SpeedReading**, Udo Gaedeke (ScanReading Ltd.)
05/2019 **Leading Yourself, Leading Others – Leadership in Scientific Environments**, Fadja Ehlail (ComAcross)
05/2019 **Leading Teams Successfully**, Corina Milek (PfO Beratungsgesellschaft)
06/2018 **Mastering the Unexpected**, Ric Oquita (impulsplus)
03/2018 **Scientific Writing**, Dr. Vera Leberecht
03/2017 **Vocal Training**, Katrin Armani
01/2017 **LaTeX with Style**, Frank Mittelbach (LaTeX3 Project Lead)
10/2016 **Conference Presentation**, Ric Oquita (impulsplus)
10/2016 **Poster-Pitch Training**, Dr. John Kluempers (textATRIUM)

Awards and Stipends

02/2021 **German IT-Security Award 2020 (2nd Prize, 60,000 EUR)**
10/2020 **PayPal Bug Bounty** for responsible disclosure of privacy vulnerability
07/2020 **Facebook Bug Bounty** for responsible disclosure of privacy vulnerability
12/2017 **CROSSING Collaboration Award 2017**
10/2012 - 09/2015 **Germany Scholarship**

Internships

08/2010 **CAIRO AG Mannheim**
07/2008 **Roche Diagnostics GmbH Mannheim**, Department of *Controlling*
01/2008 **Roche Diagnostics GmbH Mannheim**, Department of *Mechanical Engineering*

Participation in Events and Competitions

03/2021 **McKinsey & Company – Wechselwirkungen**
02/2019 **Google – Hash Code**
09/2017 **HackZurich**
02/2017 **Google – Hash Code**
12/2015 **IBM – Master the Mainframe** (Part 1 and 2 winner in D-A-CH region)

Volunteering

since 10/2017 **Germany Scholarship Commission**, CS Department, TU Darmstadt
since 02/2017 **QSL Fund Allocation Commission**, CS Department, TU Darmstadt

Scientific Service

Program Commitee Member

11/2021 **PPML 2021**
Workshop on Privacy-Preserving Machine Learning, colocated with CCS 2021
06/2021 **PrivateNLP 2021**
Workshop on Privacy in Natural Language Processing, colocated with NAACL 2021
11/2020 **PrivateNLP 2020**
Workshop on Privacy in Natural Language Processing, colocated with EMNLP 2020

External Reviewer

12/2021 **NeurIPS 2021** – Conference on Neural Information Processing Systems

11/2021 **CCS 2021** – ACM Conference on Computer and Communications Security

02/2021 **NDSS 2021** – Network and Distributed System Security Symposium

08/2020 **CRYPTO 2020** – The 40th Annual International Cryptology Conference

02/2020 **NDSS 2020** – Network and Distributed System Security Symposium

07/2019 **TIFS** – IEEE Transactions on Information Forensics & Security

07/2019 **IH&MMSEC 2019** – ACM Workshop on Information Hiding and Multimedia Security

06/2018 **ASIACCS 2018** – ACM Asia Conference on Computer and Communications Security

09/2018 **CANS 2018** International Conference on Cryptology And Network Security

04/2018 **EUROCRYPT 2018** – The 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques

12/2017 **INDOCRYPT 2017** – International Conference on Cryptology in India

11/2017 **CANS 2017** – International Conference on Cryptology And Network Security

07/2017 **ACNS 2017** – International Conference on Applied Cryptography and Network Security

04/2017 **ASIACCS 2017** – ACM Asia Conference on Computer and Communications Security

Appendices

A All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers (NDSS'21)

[HWS+21] C. HAGEN, C. WEINERT, C. SENDNER, A. DMITRIENKO, T. SCHNEIDER. **"All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers"**. In: *28. Network and Distributed System Security Symposium (NDSS'21)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2020/1119. Internet Society, 2021. CORE Rank A*. Appendix A.

https://doi.org/10.14722/ndss.2021.23159

All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers

Christoph Hagen[†], Christian Weinert[‡], Christoph Sendner[†], Alexandra Dmitrienko[†], Thomas Schneider[‡]
[†]*University of Würzburg*, Germany, {christoph.hagen,christoph.sendner,alexandra.dmitrienko}@uni-wuerzburg.de
[‡]*Technical University of Darmstadt*, Germany, {weinert,schneider}@encrypto.cs.tu-darmstadt.de

***Abstract*— Contact discovery allows users of mobile messengers to conveniently connect with people in their address book. In this work, we demonstrate that severe privacy issues exist in currently deployed contact discovery methods.**

Our study of three popular mobile messengers (WhatsApp, Signal, and Telegram) shows that, contrary to expectations, large-scale crawling attacks are (still) possible. Using an accurate database of mobile phone number prefixes and very few resources, we have queried 10 % of US mobile phone numbers for WhatsApp and 100 % for Signal. For Telegram we find that its API exposes a wide range of sensitive information, even about numbers not registered with the service. We present interesting (cross-messenger) usage statistics, which also reveal that very few users change the default privacy settings. Regarding mitigations, we propose novel techniques to significantly limit the feasibility of our crawling attacks, especially a new incremental contact discovery scheme that strictly improves over Signal's current approach.

Furthermore, we show that currently deployed hashing-based contact discovery protocols are severely broken by comparing three methods for efficient hash reversal of mobile phone numbers. For this, we also propose a significantly improved rainbow table construction for non-uniformly distributed inputs that is of independent interest.

I. Introduction

Contact discovery is a procedure run by mobile messaging applications to determine which of the contacts in the user's address book are registered with the messaging service. Newly registered users can thus conveniently and instantly start messaging existing contacts based on their phone number without the need to exchange additional information like user names, email addresses, or other identifiers[1].

Centralized messaging platforms can generally learn the social graphs of their users by observing messages exchanged between them. Current approaches to protect against this type of traffic analysis are inefficient [80], with Signal attempting to improve their service in that regard [46]. While only active users are exposed to such analyses, the contact discovery process potentially reveals *all* contacts of users to the service provider, since they must in some way be matched with the server's database. This is one of the reasons why messengers like WhatsApp might not be compliant with the European GDPR in a business context [21], [77].

[1]Some mobile applications of social networks perform contact discovery also using email addresses stored in the address book.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.23159
www.ndss-symposium.org

Cryptographic protocols for private set intersection (PSI) can perform this matching securely. Unfortunately, they are currently not efficient enough for mobile applications with billions of users [37]. Furthermore, even when deploying PSI protocols, this does not resolve all privacy issues related to contact discovery as they cannot prevent enumeration attacks, where an attacker attempts to discover which phone numbers are registered with the service.

Leaking Social Graphs. Worryingly, recent work [37] has shown that many mobile messengers (including WhatsApp) facilitate contact discovery by simply uploading *all* contacts from the user's address book[2] to the service provider and even store them on the server if no match is found [2]. The server can then notify the user about newly registered users, but can also construct the full social graph of each user. These graphs can be enriched with additional information linked to the phone numbers from other sources [12], [29], [30]. The main privacy issue here is that sensitive contact relationships can become known and could be used to scam, discriminate, or blackmail users, harm their reputation, or make them the target of an investigation. The server could also be compromised, resulting in the exposure of such sensitive information even if the provider is honest.

To alleviate these concerns, some mobile messaging applications (including Signal) implement a hashing-based contact discovery protocol, where phone numbers are transmitted to the server in hashed form [37]. Unfortunately, the low entropy of phone numbers indicates that it is most likely feasible for service providers to reverse the received hash values [50] and therefore, albeit all good intentions, there is no gain in privacy.

Crawling. Unfortunately, curious or compromised service providers are not the only threat. Malicious users or external parties might also be interested in extracting information about others. Since there are usually no noteworthy restrictions for signing up with such services, any third party can create a large number of user accounts to crawl this database for information by requesting data for (randomly) chosen phone numbers.

Such enumeration attacks cannot be fully prevented, since legitimate users must be able to query the database for contacts. In practice, rate-limiting is a well-established measure to effectively mitigate such attacks at a large scale, and one would assume that service providers apply reasonable limits to protect their platforms. As we show in § IV, this is not the case.

The simple information whether a specific phone number is registered with a certain messaging service can be sensitive in

[2]Assuming that users give the app permission to access contacts, which is very likely since otherwise they must manually enter their messenger contacts.

many ways, especially when it can be linked to a person. For example, in areas where some services are strictly forbidden, disobeying citizens can be identified and persecuted.

Comprehensive databases of phone numbers registered with a particular service can also allow attackers to perform exploitation at a larger scale. Since registering a phone number usually implies that the phone is active, such databases can be used as a reliable basis for automated sales or phishing calls. Such "robocalls" are already a massive problem in the US [79] and recent studies show that telephone scams are unexpectedly successful [78]. Two recent WhatsApp vulnerabilities, where spyware could be injected via voice calls [73] or where remote code execution was possible through specially crafted MP4 files [26], could have been used together with such a database to quickly compromise a significant number of mobile devices.

Which information can be collected with enumeration attacks depends on the service provider and the privacy settings (both in terms of which settings are chosen by the user and which are available). Examples for personal (meta) data that can commonly be extracted from a user's account include profile picture(s), nickname, status message, and the last time the user was online. In order to obtain such information, one can simply discover specific numbers, or randomly search for users [71]. By tracking such data over time, it is possible to build accurate behavior models [8], [72], [87]. Matching such information with other social networks and publicly available data sources allows third parties to build even more detailed profiles [12], [29], [30]. From a commercial perspective, such knowledge can be utilized for targeted advertisement or scams; from a personal perspective for discrimination, blackmailing, or planning a crime; and from a nation state perspective to closely monitor or persecute citizens [14]. A feature of Telegram, the possibility to determine phone numbers associated with nicknames appearing in group chats, lead to the identification of "Comrade Major" [85] and potentially endangered many Hong Kong protesters [14].

Our Contributions. We illustrate severe privacy issues that exist in currently deployed contact discovery methods by performing practical attacks both from the perspective of a curious service provider as well as malicious users.

a) Hash Reversal Attacks: Curious service providers can exploit currently deployed hashing-based contact discovery methods, which are known to be vulnerable [20], [48], [50]. We quantify the practical efforts for service providers (or an attacker who gains access to the server) for efficiently reversing hash values received from users by evaluating three approaches: (i) generating large-scale key-value stores of phone numbers and corresponding hash values for instantaneous dictionary lookups, (ii) hybrid brute-force attacks based on hashcat [74], and (iii) a novel rainbow table construction.

In particular, we compile an accurate database of worldwide mobile phone prefixes (cf. § II) and demonstrate in § III that their hashes can be reversed in just 0.1 ms amortized time per hash using a lookup database or 57 ms when brute-forcing. Our rainbow table construction incorporates the non-uniform structure of all possible phone numbers and is of independent interest. We show that one can achieve a hit rate of over 99.99 % with an amortized lookup time of 52 ms while only requiring 24 GB storage space, which improves over classical rainbow tables by more than factor 9,400x in storage.

b) Crawling Attacks: For malicious registered users and outside attackers, we demonstrate that crawling the global databases of the major mobile messaging services WhatsApp, Signal, and Telegram is feasible. Within a few weeks time, we were able to query 10 % of all US mobile phone numbers for WhatsApp and 100 % for Signal. Our attack uses very few resources: the free Hushed [1] application for registering clients with new phone numbers, a VPN subscription for rotating IP addresses, and a single laptop running multiple Android emulators. We report the rate limits and countermeasures experienced during the process, as well as other interesting findings and statistics. We also find that Telegram's API reveals sensitive personal (meta) data, most notably how many users include non-registered numbers in their contacts.

c) Mitigations: We propose a novel incremental contact discovery scheme that does not require server-side storage of client contacts (cf. § V). Our evaluation reveals that our approach enables deploying much stricter rate limits without degrading usability or privacy. In particular, the currently deployed rate-limiting by Signal can be improved by a factor of 31.6x at the cost of negligible overhead (assuming the database of registered users changes 0.1 % per day). Furthermore, we provide a comprehensive discussion on potential mitigation techniques against both hash reversal and enumeration attacks in § VI, ranging from database partitioning and selective contact permissions to limiting contact discovery to mutual contacts.

Overall, our work provides a comprehensive study of privacy issues in mobile contact discovery and the methods deployed by three popular applications with billions of users. We investigate three attack strategies for hash reversal, explore enumeration attacks at a much larger scale than previous works [30], [71], and discuss a wide range of mitigation strategies, including our novel incremental contact discovery that has the potential of real-world impact through deployment by Signal.

Outline. We first describe our approach to compile an accurate database of mobile phone numbers (§ II), which we use to demonstrate efficient reversal of phone number hashes (§ III). We also use this information to crawl WhatsApp, Signal, and Telegram, and present insights and statistics (§ IV). Regarding mitigations, we present our incremental contact discovery scheme (§ V) and discuss further techniques (§ VI). We then provide an overview of related work (§ VII) and conclude with a report on our responsible disclosure process (§ VIII).

II. Mobile Phone Number Prefix Database

In the following sections, we demonstrate privacy issues in currently deployed contact discovery methods by showing how alarmingly fast hashes of mobile phone numbers can be reversed (cf. § III) and that the database crawling of popular mobile messaging services is feasible (cf. § IV). Both attacks can be performed more efficiently with an accurate database of all possible mobile phone number prefixes[3]. Hence, we first show how such a database can be built.

A. Phone Number Structure

International phone numbers conform to a specific structure to be globally unique: Each number starts with a country

[3]Some messengers like WhatsApp and Signal also allow to register with landline phone numbers. We assume that very few users make use of this option, and also argue that gathering landline phone numbers is less attractive for attackers (e.g., when the goal is to infect smartphones with malware).

code (defined by the ITU-T standards E.123 and E.164, e.g., `+1` for the US), followed by a country-specific prefix and a subscriber number. Valid prefixes for a country are usually determined by a government body and assigned to one or more telecommunication companies. These prefixes have blocks of subscriber numbers assigned to them, from which numbers can be chosen by the provider to be handed out to customers. The length of the subscriber numbers is specific for each prefix and can be fixed or in a specified range.

In the following, we describe how an accurate list of (mobile) phone number prefixes can be compiled, including the possible length of the subscriber number. A numbering plan database is maintained by the *International Telecommunication Union* (ITU) [36] and further national numbering plans are linked therein. This database comprises more than 250 countries (including autonomous cities, city states, oversea territories, and remote island groups) and more than 9,000 providers in total. In our experiments in § IV, we focus on the US, where there are 3,794 providers (including local branches). Considering the specified minimum and maximum length of phone numbers, the prefix database allows for ≈52 trillion possible phone numbers (≈1.6 billion in the US). However, when limiting the selection to mobile numbers only, the search space is reduced to ≈758 billion (≈0.5 billion in the US).

B. Database Preprocessing

As it turned out in our experiments, some of the numbers that are supposed to be valid according to the ITU still cannot be registered with the examined messaging applications. Therefore, we perform two additional preprocessing steps.

Google's `libphonenumber` library [27] can validate phone numbers against a rule-based representation of international numbering plans and is commonly used in Android applications to filter user inputs. By filtering out invalid numbers, the amount of possible mobile phone numbers can be reduced to ≈353 billion.

Furthermore, WhatsApp performs an online validation of the numbers before registration to check, for example, whether the respective number was banned before. This allows us to check all remaining prefixes against the WhatsApp registration/login API by requesting the registration of one number for each prefix and each possible length of the subscriber number. Several more prefixes are rejected by WhatsApp for reasons like "too long" or "too short". Our final database for our further experiments thus contains up to ≈118 billion mobile phone numbers (≈0.5 billion in the US[4]). In § A we detail interesting relative differences in the amount of registrable mobile phone numbers between countries.

III. Mobile Phone Number Hash Reversal

Although the possibility of reversing phone number hashes has been acknowledged before [20], [48], [50], the severity of the problem has not been quantified. The amount of possible mobile phone numbers that we determined in § II indicates the feasibility of determining numbers based on their hash values. In the following, we show that *real-time* hash reversal is practical not only for service providers and adversaries with powerful resources, but even at a large scale using commodity hardware only.

[4] `libphonenumber` and WhatsApp reject no US mobile prefixes.

Threat Model. Here we consider the scenario where users provide hashed mobile phone numbers of their address book entries to the service provider of a mobile messaging application during contact discovery. The adversary's goal is to learn the numbers from their hashed representation. For this, we assume the adversary has full access to the hashes received by the service provider. The adversary therefore might be the service provider itself (being "curious"), an insider (e.g., an administrator of the service provider), a third party who compromised the service provider's infrastructure, or a law enforcement or intelligence agency who forces the service provider to hand out information. Importantly, we assume the adversary has no control over the users and does not tamper with the contact discovery protocol.

We compare three different approaches to reverse hashes of mobile phone numbers, each suitable for different purposes and available resources. In order to ensure comparability and uniqueness, phone numbers are processed as strings without spaces or dashes, and including their country code. Some applications add the "+"-sign as a prefix to conform to the E.164 format. In our experiments, numbers only consist of digits, but all approaches work similarly for other formats. We choose SHA-1 as our exemplary hash function, which is also used by Signal for contact discovery[5].

A. Hash Database

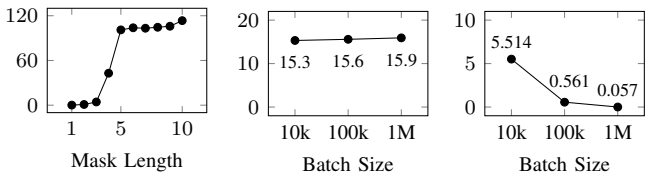
The limited amount of possible mobile phone numbers combined with the rapid increase in affordable storage capacity makes it feasible to create key-value databases of phone numbers indexed by their hashes and then to perform constant-time lookups for each given hash value. We demonstrate this by using a high-performance cluster to create an in-memory database of all 118 billion possible mobile phone numbers from § II-B (i.e., mobile phone numbers allowed by Google's `libphonenumber` and the WhatsApp registration API) paired with their SHA-1 hashes.

Benchmarks. We use one node in our cluster, consisting of 48 Intel Skylake cores at 2.3 GHz, 630 GB of RAM, and 1 TB of disk storage. We choose a Redis database due to its robustness, in-memory design, and near constant lookup-time [70]. Since one Redis instance cannot handle the required number of keys, we construct a cluster of 120 instances on our node. Populating the table requires ≈13 h in our experiments due to several bottlenecks, e.g., the interface to the Redis cluster can only be accessed through a network interface. Unfortunately, only 8 billion hashes (roughly 6.8 % of the considered number space) can fit into the RAM with our test setup. We perform batched lookups of 10,000 items, which on average take 1.0 s, resulting in an amortized lookup time of 0.1 ms.

To cover the entire mobile phone number space, a system with several Terabytes of RAM would be necessary, which makes this type of hash reversal feasible for attackers with moderate financial resources, such as large companies or nation state actors. For attackers with consumer hardware, it would also be feasible to store a full database on disk, which requires roughly 3.3 TB of storage space[6], but results in significantly higher lookup times due to disk access latencies.

[5] Signal truncates the SHA-1 output to 10 B to reduce communication overhead while still producing unique hashes for all possible phone numbers.

[6] Assuming SHA-1 hashes of 20 bytes and 64-bit encoded phone numbers.



(a) Hash rates in MH/s. (b) Total times in h. (c) Amort. times in s.

Figure 1: Brute-force benchmark results.

B. Brute-Force

Another possibility to reverse phone number hashes is to iteratively hash every element of the input domain until a matching hash is found. A popular choice for this task is the open-source tool hashcat [74], which is often used to brute-force password hashes. Hashcat can efficiently parallelize the brute-force process and additionally utilize GPUs to maximize performance. With its *hybrid* brute-forcing mode it is possible to specify *masks* that constrain the inputs according to a given structure. We use this mode to model our input space of 118 billion mobile phone numbers (cf. § II-B).

Benchmarks. We perform lookups of phone number hashes on one node of our high-performance cluster with two Intel Xeon Gold 6134 (8 physical cores at 3.2 GHz), 384 GB of RAM, and two NVIDIA Tesla P100 GPUs (16 GB of RAM each). Our setup has a theoretical rate of 9.5 GHashes/s according to the hashcat benchmark. This would allow us to search the full mobile phone number space in less than 13 seconds.

However, the true hash rate is significantly lower due to the overhead introduced by hashcat when distributing loads for processing. Since many of the prefixes have short subscriber numbers (e.g., 158,903 prefixes with length 4 digits), the overhead of distributing the masks is the bottleneck for the calculations, dropping the true hash rate to 4.3 MHashes/s for 3-digit masks (less than 0.05 % efficiency). The hash rate reaches its plateau at around 105 MHashes/s for masks larger than 4 digits (cf. Fig. 1a), which is still only 1.1 % of the theoretical hash rate.

A full search over the number space can be completed in 15.3 hours for batches of 10,000 hashes. While the total time only slightly increases with larger batch sizes (cf. Fig. 1b), the amortized lookup rate drops significantly, to only 57 ms per hash for batches of 1 million hashes (cf. Fig. 1c). Consequently, the practical results show that theoretical hash rates cannot be reached by simply deploying hashcat and that additional engineering effort would be required to optimize brute-force software for efficient phone number hash reversal.

C. Optimized Rainbow Tables

Rainbow tables are an interesting time-memory trade-off to reverse hashes (or any one-way function) from a limited input domain. Based on work from Hellman [32] and Oechslin [55], they consist of precomputed chains of plaintexts from the input domain and their corresponding hashes. These are chained together by a set of reduction functions, which map each hash back to a plaintext. By using this mapping in a deterministic chain, only the start and end of the chain must be stored to be able to search for all plaintexts in the chain. A large number of chains with random start points form a rainbow table, which can be searched by computing the chain for the given hash, and checking if the end point matches one of the entries in the table. If a match is found, then the chain can be computed from the corresponding start index to reveal the original plaintext. The length of the chains determines the time-memory trade-off: shorter chain lengths require more chains to store the same number of plaintexts, while longer chains increase the computation time for lookups. The success rate of lookups is determined by the number of chains, where special care has to be taken to limit the number of duplicate entries in the table by carefully choosing the reduction functions.

Each rainbow table is specific to the hash algorithm being used, as well as the specifications of the input domain, which determines the reduction functions. Conventional rainbow tables work by using a specific alphabet as well as a maximum input length, e.g., 8-digit ASCII numbers[7]. While they can be used to work on phone numbers as well, they are extremely inefficient for this purpose: to cover numbers conforming to the E.164 standard (up to 15 digits), the size of the input domain would be 10^{15}, requiring either huge storage capacity or extremely long chains to achieve acceptable hit rates.

By designing new reduction functions that always map a hash back into a valid phone number, we improve performance significantly. While we use our approach to optimize rainbow tables for phone numbers, our construction can also find application in other areas, e.g., advanced password cracking.

Specialized Reduction Functions. Our optimization relies on the specific structure of mobile phone numbers, which consist of a country code, a mobile prefix, and a subscriber number of a specific length (cf. § II). Conventional reduction functions simply perform a modulo operation to map each hash back to the input domain, with additional arithmetic to reduce the number of collisions in the table.

Our algorithm concatenates ranges of valid mobile phone numbers into a virtual table, which we can index with a given hash. For each prefix, we store the amount of possible subscriber numbers and the offset of the range within the table. To select a valid number, we calculate the index from the 64-bit prefix of the given hash modulo the table size and perform a binary search for the closest smaller offset to determine the corresponding mobile prefix. Subtracting the offset from the index yields the subscriber number. For example, given Tab. I and index 3,849,382, we select the prefix `+491511` and calculate the subscriber number as $3{,}849{,}382 - 110{,}000 = 3{,}739{,}382$, yielding the valid mobile phone number `+491511 3739382`.

In practice, our algorithm includes additional inputs (e.g., the current chain position) to limit the number of collisions and duplicate chains. The full specification is given in § B.

Implementation. We implement our optimized rainbow table construction based on the open-source version 1.2[8] of RainbowCrack [35]. To improve table generation and lookup performance, we add multi-threading to parts of the program via OpenMP [57]. SHA-1 hash calculations are performed using OpenSSL [58]. The table generation is modified to receive

[7]There are implementations that allow per-character alphabets [7], which is not applicable to phone numbers, since the allowed digits for each position strongly depend on the previous characters. More details are given in § C.

[8]Newer versions of RainbowCrack that support multi-threading and GPU acceleration exist, but are not open-source [68].

Country Code + Prefix	# Subscriber Numbers	Offset
+1982738	10,000	0
+172193	100,000	10,000
+491511	10,000,000	110,000
+49176	10,000,000	10,110,000

Table I: Example for selecting the next phone number from a hash value for our improved rainbow table construction.

Evaluation Criteria	Hash Database § III-A	Brute-Force § III-B	Rainbow Tables § III-C
Generation Time	13 h	–	24.5 h
RAM / Storage Requirements	$\geq$ 3.3 TB	– / –	2 GB / 24 GB
Lookup Time per 10k Batch	1 s	15.3 h	520 s
Best Amortized Time per Hash	0.1 ms	57 ms	52 ms
GPU Acceleration	✗	✓	(✓)

Table II: Comparison of phone number hash reversal methods.

the number specification as an additional parameter (a file with a list of phone number prefixes and the length of their subscriber numbers). Our open-source implementation is available at https://contact-discovery.github.io/.

Benchmarks. We generate a table of SHA-1 hashes for all registrable mobile phone numbers (118 billion numbers, cf. § II) and determine its creation time and size depending on the desired success rate for lookups, as well as lookup rates.

Our test system has an Intel Core i7-9800X with 16 physical cores and 64 GB RAM (only 2 GB are used), and can perform over 17 million hash-reduce operations per second.

We store 100 million chains of length 1,000 in each file, which results in files of 1.6 GB with a creation time of ≈98 minutes each. For a single file, we already achieve a success rate of over 50 % and an amortized lookup time of less than 26 ms for each hash when testing batches of 10,000 items. With 15 files (24 GB, created within 24.5 hours) the success rate is more than 99.99 % with an amortized lookup time of 52 ms.

In comparison, a conventional rainbow table of all 7 to 15-digit numbers has an input domain more than 9,400x larger than ours, and (with similar success rates and the same chain length) would require approximately 230 TB of storage and a creation time of more than 26 years on our test system (which is a one-time expense). The table size can be reduced by increasing the chain length, but this would result in much slower lookups.

These measurements show that our improved rainbow table construction makes large-scale hash reversal of phone numbers practical even with commodity hardware and limited financial investments. Since the created tables have a size of only a few gigabytes, they can also be easily distributed.

D. Comparison of Hash Reversal Methods

Our results for the three different approaches are summarized in Tab. II. Each approach is suitable for different application scenarios, as we discuss in the following. In § D, we discuss further optimizations for the presented methods.

A full in-memory hash database (cf. § III-A) is an option only for well-funded adversaries that require real-time reversal of hashes. It is superior to the brute-force method and rainbow tables when considering lookup latencies and total runtimes.

Brute-force cracking (cf. § III-B) is an option for a range of adversaries, from nation state actors to attackers with consumer-grade hardware, but requires non-trivial effort to perform efficiently, because publicly available tools do not perform well for phone numbers. Batching allows to significantly improve the amortized lookup rate, making brute-force cracking more suitable when a large number of hashes is to be reversed, e.g., when an attacker compromised a database.

Our optimized rainbow tables (cf. § III-C) are the approach most suited for adversaries with commodity hardware, since these tables can be calculated in reasonable time, require only a few gigabytes of storage, can be easily customized to specific countries or number ranges and types, and can reverse dozens of phone number hashes per second. It is also possible to easily share and use precomputed rainbow tables, which is done for conventional rainbow tables as well [67], despite their significantly larger size.

For other hash functions than SHA-1, we expect reversal and generation times to vary by a constant factor, depending on the computation time of the hash function [31] (except for hash databases where look-up times remain constant).

Our results show that hashing phone numbers for privacy reasons does not provide any protection, as it is easily possible to recover the original number from the hash. Thus, we strictly advise against the use of hashing-based protocols in their current form for contact discovery when users are identified by low-entropy identifiers such as phone numbers, short user names, or email addresses. In § VI-A, we discuss multiple ideas how to at least strengthen hashing-based protocols against the presented hash reversal methods.

IV. User Database Crawling

We study three popular mobile messengers to quantify the threat of enumeration attacks based on our accurate phone number database from § II-B: WhatsApp, Signal, and Telegram. All three messengers discover contacts based on phone numbers, yet differ in their implementation of the discovery service and the information exposed about registered users.

Threat Model. Here we consider an adversary who is a registered user and can query the contact discovery API of the service provider of a mobile messaging application. For each query containing a list of mobile phone numbers (e.g., in hashed form) an adversary can learn which of the provided numbers are registered with the service along with further information about the associated accounts (e.g., profile pictures). The concrete contact discovery implementation is irrelevant and it might be even based on PSI (cf. § VI-A). The adversary's goal is to check as many numbers as possible and also collect all additional information and meta data provided for the associated accounts. The adversary may control one user account or even multiple accounts, and is restricted to (ab)use the contact discovery API with well-formed queries. This implies that we assume no invasive attacks, e.g., compromising other users or the service provider's infrastructure.

A. Investigated Messengers

WhatsApp. WhatsApp is currently one of the most popular messengers in the world, with 2.0 billion users [25]. Launched in 2009, it was acquired by Facebook in 2014 for approximately 19.3 billion USD.

Signal. The Signal Messenger is an increasingly popular messenger focused on privacy. Their end-to-end-encryption protocol is also being used by other applications, such as WhatsApp, Facebook, and Skype. There are no recent statistics available regarding Signal's growth and active user base.

Telegram. Telegram is a cloud-based messenger that reported 400 million users in April 2020 [23].

B. Differences in Contact Discovery

Both WhatsApp and Telegram transmit the contacts of users in clear text to their servers (but encrypted during transit), where they are stored to allow the services to push updates (such as newly registered contacts) to the clients. WhatsApp stores phone numbers of its users in clear text on the server, while phone numbers not registered with WhatsApp are MD5-hashed with the country prefix prepended (according to court documents from 2014 [2]).

Signal does not store contacts on the server. Instead, each client periodically sends hashes of the phone numbers stored in the address book to the service, which matches them against the list of registered users and responds with the intersection.

The different procedures illustrate a trade-off between usability and privacy: the approach of WhatsApp and Telegram can provide faster updates to the user with less communication overhead, but needs to store sensitive data on the servers.

C. Test Setups

We evaluate the resistance of these three messengers against large-scale enumeration attacks with different setups.

WhatsApp. Because WhatsApp is closed source, we run the official Android application in an emulator, and use the Android UI Automator framework to control the user interface. First, we insert 60,000 new phone numbers into the address book of the device, then start the client to initiate the contact discovery. After synchronization, we can automatically extract profile information about the registered users by stepping through the contact list. New accounts are registered manually following the standard sign-up procedure with phone numbers obtained from the free Hushed [1] application.

Interestingly, if the number provided by Hushed was previously registered by another user, the WhatsApp account is "inherited", including group memberships. A non-negligible percentage of the accounts we registered had been in active use, with personal and/or group messages arriving after account takeover. This in itself presents a significant privacy risk for these users, comparable to (and possibly worse than) privacy issues associated with disposable email addresses [33]. We did not use such accounts for our crawling attempts.

Signal. The Android client of Signal is open-source, which allows us to extract the requests for registration and contact discovery, and perform them efficiently through a Python script. We register new clients manually and use the authentication tokens created upon registration to perform subsequent calls to the contact discovery API. Signal uses truncated SHA-1 hashes of the phone numbers in the contact discovery request[9]. The response from the Signal server is either an error message if the rate limit has been reached, or the hashes of the phone numbers registered with Signal.

Telegram. Interactions with the Telegram service can be made through the official library TDLib [76], which is available for many systems and programming languages. In order to create a functioning client, each project using TDLib has to be registered with Telegram to receive an authentication token, which can be done with minimal effort. We use the C++ version to perform registration and contact discovery, and to potentially download additional information about Telegram users. The registration of phone numbers is done manually by requesting a phone call to authenticate the number.

D. Ethical and Legal Considerations

We excessively query the contact discovery services of major mobile messengers, which we think is the only way to reliably estimate the success of our attacks in the real world. Similar considerations were made in previous works that evaluate attacks by crawling user data from production systems (e.g., [82]). We do not interfere with the smooth operation of the services or negatively affect other users.

In coordination with the legal department of our institution, we design the data collection process as a pipeline creating only aggregate statistics to preserve user privacy and to comply with all requirements under the European General Data Protection Regulation (GDPR) [56], especially the data minimization principle (Article 5c) and regulations of the collection of data for scientific use (Article 89). Privacy sensitive information such as profile pictures are never stored, and all data processing is performed on a dedicated local machine.

E. Rate Limits and Abuse Protection

Each messenger applies different types of protection mechanisms to prevent abuse of the contact discovery service[10].

WhatsApp. WhatsApp does not disclose how it protects against data scraping. Our experiments in September 2019 show that accounts get banned when excessively using the contact discovery service. We observe that the rate limits have a leaky bucket structure, where new requests fill a virtual bucket of a certain size, which slowly empties over time according to a specified leak rate. Once a request exceeds the currently remaining bucket size, the rate limit is reached, and the request will be denied. We estimate the bucket size to be close to 120,000 contacts, while our crawling was stable when checking 60,000 new numbers per day. There seems to be no total limit of contacts per account: some of our test accounts were able to check over 2.8 million different numbers.

Signal. According to the source code [47], the Signal servers use a leaky bucket structure. However, the parameters are not publicly available. Our measurements show that the bucket size is 50,000 contacts, while the leak rate is approximately 200,000 new numbers per day. There are no bans for clients that exceed these limits: The requests simply fail, and can be tried again later. There is no global limit for an account, as the server does not store the contacts or hashes, and thus cannot determine how many different numbers each account has already checked.

While we only use Signal's hashing-based legacy API, current Android clients also sync with the new API based on Intel SGX and compare the results. We found that the

[9] We use the legacy API; the new Intel SGX service does not use hashes.

[10] There might be additional protections not triggered by our experiments.

new API has the same rate limits as the legacy API, allowing an attacker to use both with different inputs, and thus double the effective crawling rate.

Signal clients use an additional API to download encrypted profile pictures of discovered contacts. Separate rate limits exist to protect this data, with a leaky bucket size of 4,000 and a leak rate of around 180 profiles per hour.

Telegram. The mechanism used by Telegram to limit the contact discovery process differs from WhatsApp and Signal. Telegram allows each account to add a maximum of 5,000 contacts, irrespective of the rate. Once this limit is exceeded, each account is limited to 100 new numbers per day. More requests result in a rate limit error, with multiple violations resulting in the ban of the phone number from the contact discovery service. The batch size for contact requests is 100 and performing consecutive requests with a delay of less than ≈8.3 s results in an immediate ban from the service.

In a response to the privacy issue discovered in August 2019 [14], where group members with hidden phone numbers can be identified through enumeration attacks, Telegram stated that once phone numbers are banned from contact discovery, they can only sync 5 contacts per day. We were not able to reproduce this behavior. Following our responsible disclosure, Telegram detailed additional defenses not triggered by our experiments (cf. § VIII).

F. Exposed User Data

All three messengers differ significantly regarding the amount of user data that is exposed.

WhatsApp. Users registered with WhatsApp can always be discovered by anyone through their phone number, yet the app has customizable settings for the profile picture, *About* text, and *Last Seen* information. The default for all these settings is *Everybody*, with the other options being *My Contacts* or *Nobody*. In recent Android versions it is no longer possible to save the profile picture of users through the UI, but it is possible to create screenshots through the Android Debug Bridge (ADB). The status text can be read out through the UI Automator framework by accessing the text fields in the contact list view.

Signal. The Signal messenger is primarily focused on user privacy, and thus exposes almost no information about users through the contact discovery service. The only information available about registered users is their ability to receive voice and video calls. It is also possible to retrieve the encrypted profile picture of registered users through a separate API call, if they have set any [84]. However, user name and avatar can only be decrypted if the user has consented to this explicitly for the user requesting the information and has exchanged at least one message with them [45].

Telegram. Telegram exposes a variety of information about users through the contact discovery process. It is possible to access first, last, and user name, a short bio (similar to WhatsApp's *About*), a hint when the user was last online, all profile pictures of the user (up to 100), and the number of common groups. Some of this information can be restricted to contacts only by changing the default privacy settings of the account. There is also additional management information (such as the Telegram ID), which we do not detail here.

Surprisingly, Telegram also discloses information about numbers not registered with the service through an integer

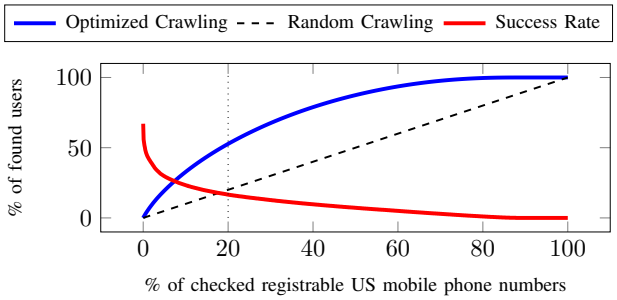


Figure 2: Optimized crawling compared to random crawling based on the non-uniform distribution of registered WhatsApp users across the US mobile phone number space.

labeled `importer_count`. According to the API documentation [75], it indicates how many registered users store a particular number in their address book, and is 0 for registered numbers[11]. Importantly, it represents the *current* state of a number, and thus decrements once users remove the number from their contacts. As such, the `importer_count` is a source of interesting meta data when keeping a specific target under surveillance. Also, when crawlers attempt to compile comprehensive databases of likely active numbers for conducting sales or phishing calls (as motivated in § I), having access to the `importer_count` increases the efficiency. And finally, numbers with non-zero values are good candidates to check on other messengers.

G. Our Evaluation Approach

We perform random lookups for mobile phone numbers in the US and collect statistics about the number of registered users, as well as the information exposed by them. The number space consists of 505.7 million mobile phone numbers (cf. § II-B). We assume that almost all users sign up for these messengers with mobile numbers, and thus exclude landline and VoIP numbers from our search space. The US numbering plan currently includes 301 3-digit area codes, which are split into 1,000 subranges of 10,000 numbers each. These subranges are handed out individually to phone companies, and only 50,573 of the 301,000 possible subranges are currently in use for mobile phone numbers. To reach our crawling targets, we select numbers evenly from all subranges. While the enumeration success rate could be increased by using telephone number lists or directories as used for telephone surveys [44], this would come at the expense of lower coverage.

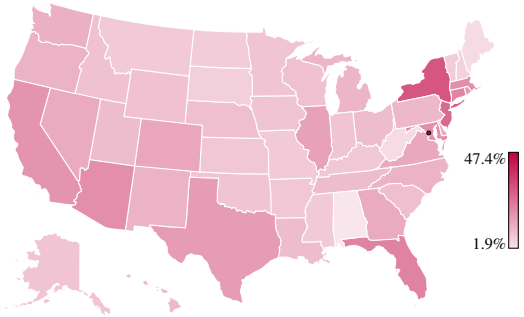
H. Our Crawling Results

The messengers have different rate limits, amount of available user information, and setup complexity. This results in different crawling speeds and number space coverage, and affects the type of statistics that can be generated.

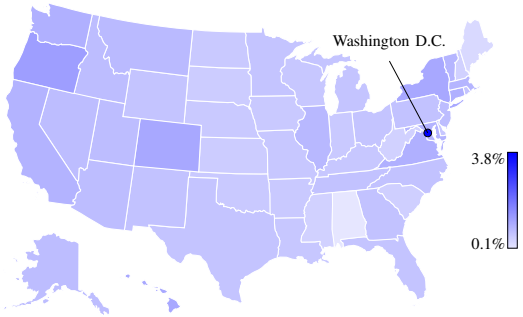
WhatsApp. For WhatsApp we use 25 accounts[12] over 34 days, each testing 60,000 numbers daily, which allows us to check 10 % of all US mobile phone numbers. For a subset of discovered users, we also check if they have public profile pictures by comparing their thumbnails to the default icon.

[11] Telegram clients use this count to suggest contacts who would benefit the most from registering.

[12] Less than 100 for Signal due to the overhead of running Android emulators.



(a) WhatsApp; the popularity is estimated based on enumerating 10 % of all possible US mobile phone numbers.



(b) Signal; Washington D.C. numbers are more than twice as likely to be registered with Signal than for any other area in the US.

Figure 3: Number of registered WhatsApp and Signal accounts of US states and Washington D.C. in relation to their population.

Messengers	WhatsApp	Signal	Telegram
Contact Discovery Method	Clear	Hashing	Clear
Rate Limits	60k / d	120k / d	5k + (100 / d)
Our Crawling Method	UI Automator	(Legacy) API	API
# US Numbers Checked	46.2 M	505.7 M	0.1 M
Coverage of US Numbers	10 %	100 %	<0.02 %
Success Rate for Random US Number	9.8 %	0.5 %	0.9 %
# US Users Found	5.0 M	2.5 M	908
# US Users (estimated)	49.6 M	2.5 M	4.6 M
Default Privacy Settings / Information Exposure			
Profile Picture	Public	Explicit Share	Public
Status	Public	–	Public
Last Online	Public	–	Public
Option to Hide Being Online	✗	✓	✓
Option to Disable Contact Discovery	✗	✗	✓

Table III: Comparison of surveyed messengers.

Users of \ also use	WhatsApp	Signal	Telegram
WhatsApp	–	2.2 %	5.1 %
Signal	42.3 %	–	8.6 %
Telegram	46.5 %	5.3 %	–

Table IV: Cross-messenger statistics for US users.

Our data shows that 5 million out of 50.5 million checked numbers are registered with WhatsApp, resulting in an average success rate of 9.8 % for enumerating random mobile phone numbers. The highest average for a single area code is 35.4 % for `718` (New York) and 35 % for `305` (Florida), while there are 209 subranges with a success rate higher than 50 % (the maximum is 67 % for a prefix in Florida). The non-uniform user distribution across the phone number space can be exploited to increase the initial success rate when enumerating entire countries, as shown in Fig. 2 for the US: with 20 % effort it is possible to discover more than 50 % of the registered users.

Extrapolating this data allows us to estimate the total number of WhatsApp accounts registered to US mobile phone numbers to be around 49.6 million. While there are no official numbers available, estimates from other sources place the number of monthly active WhatsApp users in the US at 25 million [16]. Our estimate deviates from this number, because our results include all registered numbers, not only active ones. Another statistic [17] estimates the number of US mobile phone numbers that accessed WhatsApp in 2019 at 68.1 million, which seems to be an overestimation based on our results.

For a random subset of 150,000 users we also analyzed the availability of profile pictures and *About* texts: 49.6 % have a publicly available profile picture and 89.7 % have a public *About* text. An analysis of the most popular *About* texts shows that the predefined (language-dependent) text is the most popular (77.6 %), followed by “Available” (6.71 %), and the empty string (0.81 %, including “.” and “*** no status ***”), while very few users enter custom texts.

Signal. Our script for Signal uses 100 accounts over 25 days to check all 505 million mobile phone numbers in the US. Our results show that Signal currently has 2.5 million users registered in the US, of which 82.3 % have set an encrypted user name, and 47.8 % use an encrypted profile picture. We also cross-checked with WhatsApp to see if Signal users differ in their use of public profile pictures, and found that 42.3 % of Signal users are also registered on WhatsApp (cf. Tab. IV), and 46.3 % of them have a public profile picture there. While this is slightly lower than the average for WhatsApp users (49.6 %), it is not sufficient to indicate an increased privacy-awareness of Signal’s users, at least for profile pictures.

Telegram. For Telegram we use 20 accounts running for 20 days on random US mobile phone numbers. Since Telegram’s rate limits are very strict, only 100,000 numbers were checked during that time: 0.9 % of those are registered and 41.9 % have a non-zero `importer_count`. These numbers have a higher probability than random ones to be present on other messengers, with 20.2 % of the numbers being registered with WhatsApp and 1.1 % registered with Signal, compared to the average success rates of 9.8 % and 0.9 %, respectively. Of the discovered Telegram users, 44 % of the crawled users have at least one public profile picture, with 2 % of users having more than 10 pictures available.

Summary and Comparison. An overview of the tested messengers, our crawling setup, and our most important results are given in Tab. III. Our crawling of WhatsApp, Signal, and Telegram provides insight into privacy aspects of these messengers with regard to their contact discovery service. The first notable difference is the storage of the users’ contact information, where both WhatsApp and Telegram retain this information on the server, while Signal chooses not to maintain a server-side state in order to better preserve the users’ privacy. This practice unfortunately requires significantly higher rate-limits for the contact discovery process, since all of a user’s

contacts are compared on every sync, and the server has no possibility to compare them to previously synced numbers. While Telegram uses the server-side storage of contacts to enforce strict rate limits, WhatsApp nevertheless lets individual clients check millions of numbers.

With its focus on privacy, Signal excels in exposing almost no information about registered users, apart from their phone number. In contrast, WhatsApp exposes profile pictures and the *About* text for registered numbers, and requires users to opt-out of sharing this data by changing the default settings. Our results show that only half of all US users prevent such sharing by either not uploading an image or changing the settings. Telegram behaves even worse: it allows crawling multiple images and also additional information for each user. The `importer_count` offered by its API even provides information about users not registered with the service. This can help attackers to acquire likely active numbers, which can be searched on other platforms.

Our results also show that many users are registered with multiple services (cf. Tab. IV), with 42.3 % of Signal users also being active on WhatsApp. We only found 2 out of 10,129 checked users on all three platforms (i.e., less than 0.02 %). In Fig. 3, we visualize the popularity of WhatsApp and Signal for the individual US states and Washington D.C. On average, about 10 % of residents have mobile numbers from another state [22], which may obscure these results to some extent. Interestingly, Washington D.C. numbers are more than twice as often registered on Signal than numbers from any other state, with Washington D.C. also being the region with the most non-local numbers (55 %) [22].

V. Incremental Contact Discovery

We propose a new rate-limiting scheme for contact discovery in messengers without server-side contact storage such as Signal. Setting strict limits for services without server-side contact storage is difficult, since the server cannot determine if the user's input in discovery requests changes significantly with each invocation. We named our new approach *incremental contact discovery* and shared its details with the Signal developers who consider to implement a similar approach (cf. § VIII). Our approach provides strict improvements over existing solutions, as it enables the service to enforce stricter rate limits with negligible overhead and without degrading usability or privacy.

A. Approach

Incremental contact discovery is based on the observation that the database of registered users changes only gradually over time. Similarly, the contacts of legitimate users change only slowly. Given that clients are able to store the last state for each of their contacts, they only need to query the server for changes since the last synchronization. Hence, if the server tracks database changes (new and unsubscribed users), clients who connect regularly only need to synchronize with the set of recent database changes. This enables the server to enforce stricter rate limits on the full database, which is only needed for initial synchronization, for newly added client contacts, and whenever the client fails to regularly synchronize with the set of changes. Conversely, enumeration attacks require frequent changes to the client set, and thus will quickly exceed the rate limits when syncing with the full database.

Assumptions. Based on Signal's current rate limits, we assume that each user has at most $m = 50{,}000$ contacts that are synced up to 4 times per day. This set changes slowly, i.e., only by several contacts per day. Another reasonable assumption is that the server database of registered users does not significantly change within short time periods, e.g., only 0.5 % of users join or leave the service per day (cf. § V-C).

Algorithm. The server of the service provider stores two sets of contacts: the full set S_F and the delta set S_D. S_F contains all registered users, while S_D contains only information about users that registered or unregistered within the last T_F days. Both sets, S_F and S_D, are associated with their own leaky buckets of (the same) size m, which are empty after T_F and T_D days, respectively. The server stores leaky bucket values t_F and t_D for each client, which represent the (future) points in time when the leaky buckets will be empty for requests to S_F and S_D, respectively.

A newly registered client syncs with the full set S_F to receive the current state of the user's contacts. For subsequent syncs, the client only syncs with S_D to receive recently changed contacts, provided that it synchronizes at least every T_F days. If the client is offline for a longer period of time, it can sync with S_F again, since the leaky bucket associated with it will be empty. New contacts added by the user are initially synced with S_F in order to learn their current state.

The synchronization with S_F is given in Alg. 1. It takes as inputs the server's set S_F, the maximum number of contacts m, and the associated time T_F after which the bucket will be empty. The client provides the set of contacts C_F and the server provides the client's corresponding bucket parameter t_F. The output is the set D which is the intersection of C_F with S_F, or an error, if the rate limit is exceeded.

When a client initiates a sync with S_F, the algorithm calculates t_{new}, the new (future) timestamp when the client's leaky bucket would be empty (line 1). Here, $|C_F|/m \times T_F$ represents the additional time which the bucket needs to drain. If t_{new} is further into the future than T_F (line 2), this indicates that the maximum bucket size is reached, and the request will abort with an error (line 3). Otherwise, the leaky bucket is updated for the client (line 4), and the intersection between the client set C_F and the server set S_F is returned (line 5).

The synchronization with S_D shown in Alg. 2 is quite similar. Here, the server supplies S_F, S_D, m, T_D, and t_D, and the client provides the previously synced contacts C_D. The main difference to Alg. 1 is that it outputs R_D, i.e., the requested contacts that changed (registered or unregistered) within the last T_F days together with their current state (line 5). Note that S_F is only used to check the state for contacts in S_D.

Algorithm 1 Synchronization with full set S_F

Input: S_F, m, T_F, C_F, t_F
Output: D
1: $t_{new} \leftarrow \mathbf{max}(t_F, \mathbf{current\_time}) + |C_F|/m \times T_F$
2: **if** $t_{new} > \mathbf{current\_time} + T_F$ **then**
3: | raise RateLimitExceededError
4: $t_F \leftarrow t_{new}$
5: **return** $C_F \cap S_F$

B. Implementation

We provide an open-source proof-of-concept implementation of our incremental contact discovery scheme

Algorithm 2 Synchronization with delta set S_D

Input: S_F, S_D, m, T_D, C_D, t_D
Output: R_D
1: $t_{new} \leftarrow \mathbf{max}(t_D, \mathbf{current\_time}) + |C_D|/m \times T_D$
2: **if** $t_{new} > \mathbf{current\_time} + T_D$ **then**
3: | raise RateLimitExceededError
4: $t_D \leftarrow t_{new}$
5: **return** $\{(x, x \in S_F)$ for $x \in C_D \cap S_D\}$

written in Python at https://contact-discovery.github.io/. It uses Flask [54] to provide a REST API for performing contact discovery. While not yet optimized for performance, our implementation can be useful for service providers and their developers, and in particular can facilitate integration of our idea into real-world applications.

C. Evaluation

Overhead. Our incremental contact discovery introduces only minimal server-side storage overhead, since the only additional information is the set S_D (which is small compared to S_F), as well as the additional leaky bucket states for each user. The runtime is even improved, since subsequent contact discovery requests are only compared to the smaller set S_D.

On the client side, the additional storage overhead is introduced by the need to store a timestamp of the last sync to select the appropriate set to sync with, as well as a set of previously unsynced contacts C_D.

Improvement. To evaluate our construction, we compare it to the leaky bucket approach currently deployed by Signal. Concretely, we compare the *discovery rate* of the schemes, i.e., the number of users that can be found by a single client within one day with a random lookup strategy. Rate-limiting schemes should minimize this rate for attackers without impacting usability for legitimate users. For Signal, the discovery rate is $r = s \cdot 4 \cdot 50{,}000/\mathrm{day}$, where s is the success rate for a single lookup, i.e., the ratio between registered users and all possible (mobile) phone numbers. Based on our findings in § IV-H, we assume $s = 0.5\,\%$, which results in a discovery rate of $r = 1{,}000/\mathrm{day}$ for Signal's leaky bucket approach.

For our construction, the discovery rate is the sum of the rates r_F and r_D for the buckets S_F and S_D, respectively. While r_F is calculated (similar to Signal) as $r_F = s \cdot m/T_F$, r_D is calculated as $r_D = s \cdot m \cdot c \cdot T_F/T_D$, where c is the change rate of the server database. To minimize r, we have to set $T_F = \sqrt{T_D/c}$. With Signal's parameters $s = 0.5\,\%$, $m = 50{,}000$, and $T_D = 0.25\,\mathrm{days}$, the total discovery rate for our construction therefore is $r = 1{,}000 \cdot \sqrt{c}/\mathrm{day}$, and the improvement factor is exactly $1/\sqrt{c}$.

In reality, the expected change rate depends on the popularity of the platform: Telegram saw 1.5 M new registrations per day while growing from 300 M to 400 M users [23], corresponding to a daily change rate of ≈0.5 %. WhatsApp, reporting 2 billion users in February 2020 [25] (up from 1.5 billion in January 2018 [18]), increases its userbase by an average of 0.05 % per day. Compared to Signal's rate limiting scheme, incremental contact discovery results in an improvement of 14.1x and 44.7x for Telegram's and WhatsApp's change rate, respectively (cf. Tab. V). Even at a theoretical change rate of 25 % per day, incremental discovery is twice as effective as Signal's current approach. Crawling entire countries would only be feasible for very powerful attackers, as it would require over 100k registered accounts (at $c = 0.05\,\%$) to crawl, e.g., the US in 24 hours. It should be noted that in practice the change rate c will fluctuate over time. The resulting efficiency impact of non-optimal choices for T_F is further analyzed in § E.

c (in %/d)	T_F (in d)	r (in #contacts/d)	Improvement
0.01	50.0	10.0	100.0x
0.05	22.4	22.4	44.7x
0.1	15.8	31.6	31.6x
0.5	7.1	70.7	14.1x
1.0	5.0	100.0	10.0x
2.0	3.5	141.4	7.1x

Table V: Effect of change rate c on the optimal choice for T_F, the discovery rate r for our incremental contact discovery, and the improvement compared to Signal's leaky bucket approach.

Privacy Considerations. If attackers can cover the whole number space every T_F days, it is possible to find all newly registered users and to maintain an accurate database. This is not different from today, as attackers with this capacity can sweep the full number space as well. Using the result from Alg. 2, users learn if a contact in their set has (un)registered in the last T_F days, but this information can currently also be retrieved by simply storing past discovery results.

D. Generalization

Our construction can be generalized to further decrease an attacker's efficiency. This can be achieved by using multiple sets containing the incremental changes of the server set over different time periods (e.g., one month, week, and day) such that the leak rate of S_F can be further decreased. It is even possible to use sets dynamically chosen by the service without modifying the client: each client sends its timestamp of the last sync to the service, which can be used to perform contact discovery with the appropriate set.

VI. Mitigation Techniques

We now discuss countermeasures and (mostly known) mitigation techniques for both hash reversal and enumeration attacks. We discuss further supplemental techniques in § F.

A. Hash Reversal Mitigations

Private set intersection (PSI) protocols (cf. § VII-A) can compute the intersection between the registered user database and the users' address books in a privacy-preserving manner. Thus, utilizing provably secure PSI protocols in contact discovery entirely prevents attacks where curious service providers can learn the user's social graph when receiving hashes of low-entropy contact identifiers such as phone numbers.

However, even with PSI, protocol participants can still perform enumeration attacks. Even with actively secure constructions (where privacy is still guaranteed despite arbitrary deviations from the protocol), it is possible to choose different inputs for each execution. In fact, the privacy provided by PSI interferes with efforts to detect if the respective other party replaced the majority of inputs compared to the last execution. Thus, these protocols must be combined with protections against enumeration attacks by restricting the number of protocol executions and inputs to the minimum (cf. § VI-B and § V).

Moreover, PSI protocols currently do not achieve practical performance for a very large number of users (cf. § VII-A). For example, for the current amount of about 2 billion WhatsApp users [25], each user has to initially download an encrypted and compressed database of ≈8 GiB [37]. More practical PSI designs either rely on rather unrealistic trust assumptions (e.g., non-colluding servers) or on trusted hardware [49] that provides no provable security guarantees and often suffers from side-channel vulnerabilities [9]. Hence, we discuss reasonable performance/privacy trade-offs for contact discovery next.

Database Partitioning. To reduce the communication overhead of PSI protocols to practical levels, the user database can be partitioned based on number prefixes into continents, countries, states, or even regions. This limits the service provider to learning only incomplete information about a user's social graph [37]. There are limitations to the practicality of this approach, mainly that users with diverse contacts will incur a heavy performance penalty by having to match with many partitions. For example, when partitioning based on country prefixes, a German WhatsApp user with a single contact from the US would have to additionally transfer more than 200 MiB (based on our estimates of registered US users, cf. § IV-H).[13] Also, the mere fact that a user checks contacts from a specific country might be privacy-sensitive.

Strengthened Hashing-based Protocols. Given the current scalability issues of PSI protocols, a first step could be to patch the currently deployed hashing-based protocols. One could introduce a global salt for such protocols to prevent reusable rainbow tables (cf. § III-C). Rotating the salt in short intervals also makes hash databases (cf. § III-A) less attractive.

Another alternative is to increase the calculation time of each hash, either by performing multiple rounds of the hash function or by using hash functions like bcrypt [66] or Argon2 [6], which are specifically designed to resist brute-force attacks. Existing benchmarks show that with bcrypt only 2.9 kHashes/s and with Argon2 only 2.6 Hashes/s can be computed on a GPU compared to 794.6 MHashes/s with SHA-1 [31].

These measures will not be sufficient against very powerful adversaries, but can at least increase the costs of hash reversal attacks by a factor of even millions. However, the performance penalty will also affect clients when hashing their contacts, as well as the server, when updating the database.

Alternative Identifiers. It should be possible for privacy-concerned users to provide another form of identifier (e.g., a user name or email address, as is the standard for social networks) instead of their phone number. This increases the search space for an attacker and also improves resistance of hashes against reversal. Especially random or user-chosen identifiers with high entropy would offer better protection. However, this requires to share additional data when exchanging contact information and therefore reduces usability. Signal nevertheless plans to introduce alternative identifiers [51].

Selective Contact Permissions. iOS and Android require apps to ask for permission to access the user's address book, which is currently an all or nothing choice. Mobile operating systems could implement a functionality in their address book apps to allow users to declare certain contacts as "sensitive" or "private", e.g., via a simple check box. Mobile messengers then are not able to access such protected contacts and therefore cannot leak them to the service provider.

Also the existing groups in the address book could be extended for this, e.g., declare the group of health-related contacts as sensitive and do not use them for contact discovery. There already exist wrapper apps for specific messengers with similar functionality (e.g., WhatsBox [3] for WhatsApp), but a system-wide option would be preferable.

Furthermore, users may hide contacts they deem sensitive (e.g., doctors) by not storing them in the phone's address book if messengers have access permissions. Alternatively, users can revoke access permissions for such applications.

B. Crawling Mitigations

In the following, we discuss several possible mitigation strategies that have the potential to increase resilience against crawling attacks. Furthermore, since many messenger apps give users the possibility to add additional information to their profile, we also discuss countermeasures that can prevent, or at least limit, the exposure of sensitive private information through the scraping of user profiles.

Stricter Rate Limits. Rate limits are a trade-off between user experience and protection of the service. If set too low, users with no malicious intent but unusual usage patterns (e.g., a large number of contacts) will exceed these limits and suffer from a bad user experience. This is especially likely for services with a large and diverse user base.

However, we argue that private users have no more than 10,000 contacts in their address book (Signal states similar numbers [37] and Google's contact management service limits the maximum number to 25,000 [28]). Therefore, the contact discovery service should not allow syncing more numbers than in this order of magnitude at any point in time. Exceptions could be made for businesses, non-profit organizations, or celebrities after performing extended validation.

We furthermore argue that private users do not change many of their contacts frequently. The operators of Writethat.name observed that even professional users have only about 250 new contacts per year [83]. Therefore, service providers could penalize users when detecting frequent contact changes. Additional total limits for the number of contacts can detect accounts crawling at slow rates.

Facebook (WhatsApp's parent company) informed us during responsible disclosure that they see legitimate use cases where users synchronize more contacts (e.g., enterprises with 200,000 contacts)[14]. We recommend to handle such business customers differently than private users. In response to our findings showing that data scraping is currently possible even at a country level scale (cf. § IV), Facebook informed us that they have improved WhatsApp's contact synchronization feature to detect such attacks much earlier (cf. § VIII).

Limiting Exposure of Sensitive Information. Since preventing enumeration attacks entirely is impossible, the information collected about users through this process should be kept minimal. While Signal behaves exemplarily and reveals no public profile pictures or status information, WhatsApp

[13] The PSI protocols of [37] initially transfer 4.19 MB per 1 M users.

[14] This definition of "legitimate" is interesting, since WhatsApp's terms of service prohibit *non-personal* use of their services [81].

and Telegram should set corresponding default settings. Furthermore, users themselves may take actions to protect themselves from exposure of private information by thinking carefully what information to include into public fields, such as profile pictures and status text, and checking whether there are privacy settings that can limit the visibility of this information.

Mutual Contacts. Mobile messengers could offer a setting for users to let them only be discovered on the service by contacts in their address book to prevent third parties from obtaining any information about them.

VII. Related work

We review related work from four research domains: PSI protocols, enumeration attacks, user tracking, and hash reversal.

A. Private Set Intersection (PSI)

PSI protocols can be used for mobile private contact discovery to hinder hash reversal attacks (cf. § III). Most PSI protocols consider a scenario where the input sets of both parties have roughly the same size (e.g., [43], [60], [61], [62], [63], [64]). However, in contact discovery, the provider has orders of magnitude more entries in the server database than users have contacts in their address book. Thus, there has been research on *unbalanced* PSI protocols, where the input set of one party is much larger than the other [10], [11], [37], [41].

Today's best known protocols [37] also provide efficient implementations with reasonable runtimes on modern smartphones. Unfortunately, their limitation is the amount of data that needs to be transferred to the client in order to obtain an encrypted representation of the server's database: for 2^{28} registered users (the estimated number of active users on Telegram [15]) it is necessary to transfer ≈1 GiB, for 2^{31} registered users (a bit more than the estimated number of users on WhatsApp [15]) even ≈8 GiB are necessary. Moreover, even PSI protocols cannot prevent enumeration attacks, as discussed in § VI-A.

The Signal developers concluded that current PSI protocols are not practical for deployment [49], and also argue that the required non-collusion assumption for more efficient solutions with multiple servers [37] is unrealistic. Instead, they introduced a beta version [49] that utilizes Intel Software Guard Extensions (SGX) for securely performing contact discovery in a trusted execution environment. However, Intel SGX provides no provable security guarantees and there have been many severe attacks (most notably "Foreshadow" [9]). Given the scope of such attacks and that fixes often require hardware changes, the Intel SGX-based contact discovery service is less secure than cryptographic PSI protocols.

B. Enumeration Attacks

Popular applications for enumeration attacks include, e.g., finding vulnerable devices by scanning all IPv4 addresses and ports. In the following, we focus on such attacks on social networks and mobile messengers.

For eight popular social networks, Balduzzi et al. [4] fed about 10 million cleverly generated email addresses into the search interface, allowing them to identify 1.2 million user profiles without experiencing any form of countermeasure. After crawling these profiles with methods similar to [5], they correlated the profiles from different networks to obtain a combined profile that in many cases contained friend lists, location information, and sexual preferences. Upon the responsible disclosure of their findings, Facebook and XING quickly established reasonable rate limits for search queries. We hope for similar deployment of countermeasures by responsively disclosing our findings on mobile messengers (cf. § VIII).

Schrittwieser et al. [53], [71] were the first to investigate enumeration attacks on mobile messengers, including WhatsApp. For the area code of San Diego, they automatically tested 10 million numbers within 2.5 hours without noticing severe limitations. Since then, service providers established at least some countermeasures. We revisit enumeration attacks at a substantially larger scale (cf. § IV) and demonstrate that the currently deployed countermeasures are insufficient to prevent large-scale enumeration attacks.

For the Korean messenger KakaoTalk, enumeration attacks were demonstrated in [38], [39]. The authors automatically collected ≈50,000 user profiles by enumerating 100,000 number sequences that could potentially be phone numbers. They discovered a method to obtain the user names associated with these profiles and found that ≈70 % of users chose their real name (or at least a name that could be a real name), allowing identification of many users. As countermeasures, the authors propose the detection of certain known misuse patterns as well as anomaly detection for repeated queries. In contrast, in § IV we automatically perform enumeration attacks at a much larger scale on popular messaging applications used world-wide. By testing only valid mobile phone numbers, we increase the efficiency of our attacks. We propose further mitigations in § VI.

In [12], the authors describe a simple Android-based system to automatically conduct enumeration attacks for different mobile messengers by triggering and recording API calls via the debug bridge. In their evaluation, they enumerate 100,000 Chinese numbers for WeChat and correlate the results with other messengers. We perform evaluations of different messengers at a larger scale, also assessing currently deployed countermeasures against enumeration attacks (cf. § IV).

Gupta et al. [29], [30] obtained personal information from reverse-lookup services, which they correlated with public profiles on social networks like Facebook, in order to then run personalized phishing attacks on messengers like WhatsApp. From about 1 million enumerated Indian numbers, they were able to target about 250,000 users across different platforms.

Enumeration attacks were also used to automatically harvest Facebook profiles associated with phone numbers even when the numbers are hidden in the profiles [40]. The authors experienced rather strict countermeasures that limit the number of possible queries to 300 before a "security check" in form of a CAPTCHA is triggered. By automatically creating many fake accounts and setting appropriately slow crawling rates, it was still possible to test around 200,000 Californian and Korean phone numbers within 15 days, leading to a success rate of 12 % and 25 %, respectively. While acquiring phone numbers is more cumbersome than generating email addresses, we nevertheless report much faster enumeration attacks that harvest profiles of mobile messenger users (cf. § IV).

In 2017, Loran Kloeze developed the Chrome extension "WhatsAllApp" that allows to misuse WhatsApp's web interface for enumeration attacks and collecting profile pictures, display names, and status information [42]. After disclosing his approach, Facebook pointed out (non-default) privacy settings

available to the user to hide this information, and stated that WhatsApp detects abuse based on measures that identify and block data scraping [19]. In § IV, we investigate the effectiveness of their measures and find that we can perform attacks at a country-level scale, even with few resources. We also observe that few users change the default settings.

There exist other open-source projects that enable automated crawling of WhatsApp users and extracting personal information, e.g., [24], [65]. However, frequent changes of the WhatsApp API and code often break these tools, which are mostly abandoned after some time, or cease operation after receiving legal threats [34].

C. User Tracking

In 2014, Buchenscheit et al. [8] conducted a user study where they tracked online status of participants for one month, which allowed them to infer much about the participants' daily routines and conversations (w.r.t. duration and chat partners). Other user studies report the "Last Seen" feature as the users' biggest privacy concern in WhatsApp [13], [69].

Researchers also monitored the online status of 1,000 randomly selected users from different countries for 9 months [72]. They published statistics on the observed behavior w.r.t. the average usage time per day and the usage throughout the day. Despite the clearly anomalous usage patterns of the monitoring, the authors did not experience any countermeasures.

"WhatsSpy" is an open-source tool that monitors the online status, profile pictures, and status messages of selected numbers—provided the default privacy options are set [87]. It abuses the fact that WhatsApp indicates whether a user is online [88], even when the "Last Seen" feature is disabled. The tool was discontinued in 2016 to prevent low-level abuse [89], since the developer found more than 45,000 active installations and companies trying to use the prototype commercially.

In this context, our user database crawling attacks could be used to efficiently find new users to track and our discovery of Telegram's `importer_count` label gives even more monitoring possibilities (cf. § IV).

D. Hash Reversal

Reversing hashes is mostly used for "recovering" passwords, which are commonly stored only in hashed form. Various hash reversal tools exist, either relying on brute-forcing [59], [74] or rainbow tables [68]. The practice of adding a unique salt to each hash makes reversal hard at a large scale, but is not suitable for contact discovery [37], [48]. In contrast, our mitigation proposed in § VI-A uses a *global* salt.

It is well known that hashing of personally identifiable information (PII), including phone numbers, is not sufficient due to the small pre-image space [20], [48]. The PSI literature therefore has proposed many secure alternatives for matching PII, which are currently orders of magnitudes slower than insecure hashing-based protocols (cf. § VII-A).

In [50], the authors show that the specific structure of PII makes attacks much easier in practice. Regarding phone numbers, they give an upper bound of 811 trillion possible numbers world-wide, for which brute-forcing takes around 11 days assuming SHA-256 hashes and a hash rate of 844 MH/s. For specific countries, they also run experiments showing that reversing an MD5 or SHA-256 hash for a German phone number takes at most 2.5 hours. In § II, we give much more accurate estimations for the amount of possible (mobile) phone numbers and show in § III that using novel techniques and optimizations, hash reversal is much faster and can even be performed on-the-fly.

VIII. Conclusion

Mobile contact discovery is a challenging topic for privacy researchers in many aspects. In this paper, we took an attacker's perspective and scrutinized currently deployed contact discovery services of three popular mobile messengers: WhatsApp, Signal, and Telegram. We revisited known attacks and using novel techniques we quantified the efforts required for curious service providers and malicious users to collect sensitive user data at a large scale. Shockingly, we were able to demonstrate that still almost nothing prevents even resource-constraint attackers from collecting data of billions of users that can be abused for various purposes. While we proposed several technical mitigations for service providers to prevent such attacks in the future, currently the most effective protection measure for users is to revise the existing privacy settings. Thus, we advocate to raise awareness among regular users about the seriousness of privacy issues in mobile messengers and educate them about the precautions they can take right now.

Responsible Disclosure. In our paper, we demonstrate methods that allow to invade the privacy of billions of mobile messenger users by using only very few resources. We therefore initiated the official responsible disclosure process with all messengers we investigated (WhatsApp, Signal, and Telegram) before the paper submission and shared our findings to prevent exploitation by maleficent imitators.

Signal acknowledged the issue of enumeration attacks as not fully preventable, yet nevertheless adjusted their rate limits in the weeks following our disclosure and implemented further defenses against crawling. Facebook acknowledged and rewarded our findings as part of their bug bounty program, and has deployed improved defenses for WhatsApp's contact synchronization. Telegram responded to our responsible disclosure by elaborating on additional data scraping countermeasures beyond the rate limits detected by us. They are allegedly triggered when attackers use existing databases of active phone numbers and higher conversion rates than ours occur. In such cases, contact discovery is stopped after 20 to 100 matches, instead of 5,000 as measured by us.

Ethical Considerations. The experiments in this work were conducted in coordination with the ethical and legal departments of our institution. Special care was taken to ensure the privacy of the affected users, as detailed in § IV-D.

Acknowledgments

We thank Lukas Nothelfer, Florian Plesker, Oliver Schick, and Sebastian Schindler for their invaluable help with the implementation of our attacks.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 850990 PSOTI). It was supported by the DFG as part of project E4 within the CRC 1119 CROSSING and project A.1 within the RTG 2050 "Privacy and Trust for Mobile Users", and by the BMBF and HMWK within CRISP and ATHENE.

REFERENCES

[1] Affinityclick, "Hushed - Private Phone Numbers, Talk and Text," 2019. [Online]. Available: https://hushed.com/

[2] P. Aftab, "Findings under the Personal Information Protection and Electronic Documents Act (PIPEDA)," 2014. [Online]. Available: https://parryaftab.blogspot.com/2014/03/what-does-whatsapp-collect-that.html

[3] Backes SRT, "WhatsBox - GDPR Compliant WhatsApp," 2013. [Online]. Available: https://www.backes-srt.com/en/solutions-2/whatsbox/

[4] M. Balduzzi, C. Platzer, T. Holz, E. Kirda, D. Balzarotti, and C. Kruegel, "Abusing Social Networks for Automated User Profiling," in *Recent Advances in Intrusion Detection (RAID)*. Springer, 2010, pp. 422–441. [Online]. Available: https://doi.org/10.1007/978-3-642-15512-3_22

[5] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, "All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks," in *International Conference on World Wide Web (WWW)*. ACM, 2009, pp. 551–560. [Online]. Available: https://doi.org/10.1145/1526709.1526784

[6] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications," in *EuroS&P*. IEEE, 2016, pp. 292–302. [Online]. Available: https://doi.org/10.1109/EuroSP.2016.31

[7] BitWeasil, "Cryptohaze," 2012. [Online]. Available: http://www.cryptohaze.com

[8] A. Buchenscheit, B. Könings, A. Neubert, F. Schaub, M. Schneider, and F. Kargl, "Privacy Implications of Presence Sharing in Mobile Messaging Applications," in *International Conference on Mobile and Ubiquitous Multimedia*. ACM, 2014, pp. 20–29. [Online]. Available: https://doi.org/10.1145/2677972.2677980

[9] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution," in *USENIX Security*. USENIX, 2018, pp. 991–1008. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-van_bulck.pdf

[10] H. Chen, Z. Huang, K. Laine, and P. Rindal, "Labeled PSI from Fully Homomorphic Encryption with Malicious Security," in *CCS*. ACM, 2018, pp. 1223–1237. [Online]. Available: https://doi.org/10.1145/3243734.3243836

[11] H. Chen, K. Laine, and P. Rindal, "Fast Private Set Intersection from Homomorphic Encryption," in *CCS*. ACM, 2017, pp. 1243–1255. [Online]. Available: https://doi.org/10.1145/3133956.3134061

[12] Y. Cheng, L. Ying, S. Jiao, P. Su, and D. Feng, "Bind Your Phone Number with Caution: Automated User Profiling Through Address Book Matching on Smartphone," in *ASIACCS*. ACM, 2013, pp. 335–340. [Online]. Available: https://doi.org/10.1145/2484313.2484356

[13] K. Church and R. de Oliveira, "What's Up with WhatsApp? Comparing Mobile Instant Messaging Behaviors with Traditional SMS," in *Human-Computer Interaction with Mobile Devices and Services (MobileHCI)*. ACM, 2013, pp. 352–361. [Online]. Available: https://doi.org/10.1145/2493190.2493225

[14] C. Cimpanu, "Hong Kong Protesters Warn of Telegram Feature that can Disclose Their Identities," 2019. [Online]. Available: https://www.zdnet.com/article/hong-kong-protesters-warn-of-telegram-feature-that-can-disclose-their-identities/

[15] J. Clement, "Most Popular Global Mobile Messenger Apps," 2019. [Online]. Available: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps

[16] ——, "Most Popular Mobile Messaging Apps in the United States as of June 2019," 2019. [Online]. Available: https://www.statista.com/statistics/350461/mobile-messenger-app-usage-usa/

[17] ——, "Number of WhatsApp Users in the United States from 2019 to 2023," 2019. [Online]. Available: https://www.statista.com/statistics/558290/number-of-whatsapp-users-usa/

[18] J. Constine, "WhatsApp hits 1.5 billion monthly users. $19B? Not so bad." 2018. [Online]. Available: https://techcrunch.com/2018/01/31/whatsapp-hits-1-5-billion-monthly-users-19b-not-so-bad/

[19] J. Cox, "Collecting Huge Amounts of Data with WhatsApp," 2017. [Online]. Available: https://www.vice.com/en_us/article/gvzw5x/secure-messaging-app-wire-stores-everyone-youve-ever-contacted-in-plain-text

[20] L. Demir, A. Kumar, M. Cunche, and C. Lauradoux, "The Pitfalls of Hashing for Privacy," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 1, pp. 551–565, 2018. [Online]. Available: https://doi.org/10.1109/COMST.2017.2747598

[21] Deutsche Welle, "New EU Data Law Forces Firms to Ban WhatsApp, Snapchat from Phones," 2019. [Online]. Available: https://www.dw.com/en/new-eu-data-law-forces-firms-to-ban-whatsapp-snapchat-from-phones/a-44076861

[22] M. Dost and K. McGeeney, "Moving Without Changing Your Cellphone Number: A Predicament for Pollsters," 2016. [Online]. Available: https://www.pewresearch.org/methods/2016/08/01/moving-without-changing-your-cellphone-number-a-predicament-for-pollsters/

[23] P. Durov, "400 Million Users, 20,000 Stickers, Quizzes 2.0 and 400K EUR for Creators of Educational Tests," 2020. [Online]. Available: https://telegram.org/blog/400-million

[24] J. Estrada, "WhatsApp Scraping," 2019. [Online]. Available: https://github.com/JMGama/WhatsApp-Scraping

[25] I. Facebook, "Two Billion Users — Connecting the World Privately," 2020. [Online]. Available: https://about.fb.com/news/2020/02/two-billion-users/

[26] Forbes, "New WhatsApp Threat Confirmed: Android And iOS Users At Risk From Malicious Video Files," 2019. [Online]. Available: https://www.forbes.com/sites/zakdoffman/2019/11/16/new-whatsapp-threat-confirmed-android-and-ios-users-at-risk-from-malicious-video-files/

[27] Google, "Google's Common Java, C++ and JavaScript Library for Parsing, Formatting, and Validating International Phone Numbers," 2019. [Online]. Available: https://github.com/google/libphonenumber

[28] Google, "I'm getting a Contacts error - Contacts Help," 2019. [Online]. Available: https://support.google.com/contacts/answer/148779

[29] S. Gupta, "Emerging Threats Abusing Phone Numbers Exploiting Cross-Platform Features," in *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2016, pp. 1339–1341. [Online]. Available: https://doi.org/10.1109/ASONAM.2016.7752410

[30] S. Gupta, P. Gupta, M. Ahamad, and P. Kumaraguru, "Exploiting Phone Numbers and Cross-Application Features in Targeted Mobile Attacks," in *Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM@CCS)*. ACM, 2016, pp. 73–82. [Online]. Available: https://doi.org/10.1145/2994459.2994471

[31] G. Hatzivasilis, "Password-Hashing Status," *Cryptography*, vol. 1, no. 2, p. 10, 2017. [Online]. Available: https://doi.org/10.3390/cryptography1020010

[32] M. Hellman, "A Cryptanalytic Time-Memory Trade-Off," *Transactions on Information Theory*, vol. 26, no. 4, pp. 401–406, 1980. [Online]. Available: https://doi.org/10.1109/TIT.1980.1056220

[33] H. Hu, P. Peng, and G. Wang, "Characterizing Pixel Tracking through the Lens of Disposable Email Services," in *S&P*. IEEE, 2019, pp. 365–379. [Online]. Available: https://doi.org/10.1109/SP.2019.00033

[34] A. Hubail, "Interface to WhatsApp Messenger—Fed up with the F**king Legal Threats," 2015. [Online]. Available: https://github.com/venomous0x/WhatsAPI

[35] inAudible-NG, "RainbowCrack-NG: Free and Open-Source Software to Generate and Use Rainbow Tables," 2017. [Online]. Available: https://github.com/inAudible-NG/RainbowCrack-NG

[36] ITU Telecommunication Standardization Sector, "National Numbering Plans," 2019. [Online]. Available: https://www.itu.int/oth/T0202.aspx?parent=T0202

[37] D. Kales, C. Rechberger, M. Senker, T. Schneider, and C. Weinert, "Mobile Private Contact Discovery at Scale," in *USENIX Security*. USENIX, 2019, pp. 1447–1464. [Online]. Available: https://ia.cr/2019/517

[38] E. Kim, K. Park, H. Kim, and J. Song, "I've Got Your Number: - Harvesting Users' Personal Data via Contacts Sync for the KakaoTalk Messenger," in *Workshop on Information Security Applications (WISA)*. Springer, 2014, pp. 55–67. [Online]. Available: https://doi.org/10.1007/978-3-319-15087-1_5

[39] ——, "Design and Analysis of Enumeration Attacks on Finding Friends with Phone Numbers: A Case Study with KakaoTalk," *Computers & Security*, vol. 52, pp. 267–275, 2015. [Online]. Available: https://doi.org/10.1016/j.cose.2015.04.008

[40] J. Kim, K. Kim, J. Cho, H. Kim, and S. Schrittwieser, "Hello, Facebook! Here Is the Stalkers' Paradise!: Design and Analysis of Enumeration Attack Using Phone Numbers on Facebook," in *Information Security Practice and Experience*. Springer, 2017, pp. 663–677. [Online]. Available: https://doi.org/10.1007/978-3-319-72359-4_41

[41] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas, "Private Set Intersection for Unequal Set Sizes with Mobile Applications," *PoPETs*, vol. 2017, no. 4, pp. 177–197, 2017. [Online]. Available: https://doi.org/10.1515/popets-2017-0044

[42] L. Kloeze, "Collecting Huge Amounts of Data with WhatsApp," 2017. [Online]. Available: https://www.lorankloeze.nl/2017/05/07/collecting-huge-amounts-of-data-with-whatsapp/

[43] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient Batched Oblivious PRF with Applications to Private Set Intersection," in *CCS*. ACM, 2016, pp. 818–829. [Online]. Available: https://doi.org/10.1145/2976749.2978381

[44] J. M. Lepkowski, "Telephone Sampling: Frames and Selection Techniques," in *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 1585–1586. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_96

[45] J. Lund, "Encrypted Profiles for Signal Now in Public Beta," 2017. [Online]. Available: https://signal.org/blog/signal-profiles-beta/

[46] ——, "Technology Preview: Sealed Sender for Signal," 2018. [Online]. Available: https://signal.org/blog/sealed-sender/

[47] ——, "Signal-Server," 2019. [Online]. Available: https://github.com/signalapp/Signal-Server

[48] M. Marlinspike, "The Difficulty Of Private Contact Discovery," 2014. [Online]. Available: https://signal.org/blog/contact-discovery/

[49] ——, "Technology Preview: Private Contact Discovery for Signal," 2017. [Online]. Available: https://signal.org/blog/private-contact-discovery

[50] M. Marx, E. Zimmer, T. Mueller, M. Blochberger, and H. Federrath, "Hashing of Personally Identifiable Information is Not Sufficient," in *Sicherheit*. Gesellschaft für Informatik e.V., 2018, pp. 55–68. [Online]. Available: https://doi.org/10.18420/sicherheit2018_04

[51] S. Messenger, "Introducing Signal PINs," 2020. [Online]. Available: https://signal.org/blog/signal-pins/

[52] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, "Re: CAPTCHAs-Understanding CAPTCHA-Solving Services in an Economic Context," in *USENIX Security*. USENIX, 2010, pp. 435–462. [Online]. Available: http://www.usenix.org/events/sec10/tech/full_papers/Motoyama.pdf

[53] R. Mueller, S. Schrittwieser, P. Frühwirt, P. Kieseberg, and E. R. Weippl, "What's New with WhatsApp & Co.? Revisiting the Security of Smartphone Messaging Applications," in *Information Integration and Web-based Applications & Services*. ACM, 2014, pp. 142–151. [Online]. Available: https://doi.org/10.1145/2684200.2684328

[54] A. Mönnich, "Flask," 2019. [Online]. Available: https://palletsprojects.com/p/flask

[55] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," in *CRYPTO*. Springer, 2003, pp. 617–630. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_36

[56] Official Journal of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council," 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN

[57] OpenMP. (2019) The OpenMP API Specification for Parallel Programming. [Online]. Available: https://www.openmp.org

[58] OpenSSL Software Foundation, "OpenSSL: Cryptography and SSL/TLS Toolkit," 2018. [Online]. Available: https://www.openssl.org

[59] Openwall, "John the Ripper Password Cracker," 2019. [Online]. Available: https://www.openwall.com/john/

[60] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension," in *CRYPTO*. Springer, 2019, pp. 401–431. [Online]. Available: https://doi.org/10.1007/978-3-030-26954-8_13

[61] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private Set Intersection Using Permutation-based Hashing," in *USENIX Security*. USENIX, 2015, pp. 515–530. [Online]. Available: http://ia.cr/2015/634

[62] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, "Efficient Circuit-Based PSI via Cuckoo Hashing," in *EUROCRYPT*. Springer, 2018, pp. 125–157. [Online]. Available: https://doi.org/10.1007/978-3-319-78372-7_5

[63] B. Pinkas, T. Schneider, and M. Zohner, "Faster Private Set Intersection Based on OT Extension," in *USENIX Security*. USENIX, 2014, pp. 797–812. [Online]. Available: http://ia.cr/2014/447

[64] ——, "Scalable Private Set Intersection Based on OT Extension," *Transactions on Privacy and Security (TOPS)*, vol. 21, no. 2, pp. 7:1–7:35, 2018. [Online]. Available: https://doi.org/10.1145/3154794

[65] S. PJ, "WhatsApp Crawler," 2017. [Online]. Available: https://gitlab.com/jishnutp/whatsapp-crawler

[66] N. Provos and D. Mazières, "A Future-Adaptable Password Scheme," in *USENIX Annual Technical Conference (ATC)*. USENIX, 1999, pp. 81–91. [Online]. Available: https://www.usenix.org/legacy/events/usenix99/provos/provos.pdf

[67] RainbowCrack Project, "List of Rainbow Tables," 2019. [Online]. Available: http://project-rainbowcrack.com/table.htm

[68] ——, "RainbowCrack," 2019. [Online]. Available: http://project-rainbowcrack.com/

[69] Y. Rashidi, K. Vaniea, and L. J. Camp, "Understanding Saudis' Privacy Concerns When Using WhatsApp," in *Workshop on Usable Security (USEC)*. Internet Society, 2016. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/understanding-saudis-privacy-concerns-when-using-whatsapp.pdf

[70] S. Sanfilippo, "Redis Commands - GET," 2019. [Online]. Available: https://redis.io/commands/get

[71] S. Schrittwieser, P. Frühwirt, P. Kieseberg, M. Leithner, M. Mulazzani, M. Huber, and E. R. Weippl, "Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications," in *NDSS*. Internet Society, 2012. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/07_1.pdf

[72] Security Research Group FAU Erlangen-Nürnberg, "Online Status Monitor," 2014. [Online]. Available: https://onlinestatusmonitor.com/

[73] M. Srivastava, "WhatsApp Voice Calls Used to Inject Israeli Spyware on Phones," 2019. [Online]. Available: https://www.ft.com/content/4da1117e-756c-11e9-be7d-6d846537acab

[74] J. Steube and G. Gristina, "hashcat - World's Fastest and Most Advanced Password Recovery Utility," 2019. [Online]. Available: https://hashcat.net/

[75] Telegram, "TDLib: importedContacts Class Reference," 2019. [Online]. Available: https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1imported_contacts.html

[76] ——, "Telegram Database Library," 2019. [Online]. Available: https://core.telegram.org/tdlib

[77] Tom Slack, "Is WhatsApp in Breach of the GDPR? A Lawyer's View," 2019. [Online]. Available: https://guild.co/blog/is-whatsapp-in-breach-of-the-gdpr-a-lawyers-view/

[78] H. Tu, A. Doupé, Z. Zhao, and G. Ahn, "Users Really Do Answer Telephone Scams," in *USENIX Security*. USENIX, 2019, pp. 1327–1340. [Online]. Available: https://www.usenix.org/system/files/sec19-tu.pdf

[79] L. Vaas, "Robocalls Now Flooding US Phones with 200m Calls per Day," 2019. [Online]. Available: https://nakedsecurity.sophos.com/2019/09/17/robocalls-now-flooding-us-phones-with-200m-calls-per-day/

[80] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis," in *Symposium on Operating Systems Principles (SOSP)*. ACM, 2015, pp. 137–152. [Online]. Available: https://doi.org/10.1145/2815400.2815417

[81] WhatsApp Inc., "WhatsApp Legal Info," 2019. [Online]. Available: https://www.whatsapp.com/legal?eea=0#terms-of-service

[82] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A Practical Attack to De-anonymize Social Network Users," in *S&P*. IEEE, 2010, pp. 223–238. [Online]. Available: https://doi.org/10.1109/SP.2010.21

[83] WriteThat.Name, "Your Address Book Automagically Updated," 2013. [Online]. Available: http://writethat.name/

[84] x0rz, "A Look Into Signal's Encrypted Profiles," 2018. [Online]. Available: https://blog.0day.rocks/a-look-into-signals-encrypted-profiles-5491908186c1

[85] L. Yapparova and A. Kovalev, "Comrade Major," 2019. [Online]. Available: https://meduza.io/en/feature/2019/08/11/comrade-major

[86] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang, "Yet Another Text CAPTCHA Solver: A Generative Adversarial Network Based Approach," in *CCS*. ACM, 2018, pp. 332–348. [Online]. Available: https://doi.org/10.1145/3243734.3243754

[87] M. Zweerink, "WhatsApp Privacy is Broken!" 2015. [Online]. Available: https://maikel.pro/blog/en-whatsapp-privacy-options-are-illusions/

[88] ——, "WhatsApp Privacy Problem Explained in Detail," 2015. [Online]. Available: https://maikel.pro/blog/en-whatsapp-privacy-problem-explained-in-detail/

[89] ——, "PoC WhatsSpy Public Support Ending Today," 2016. [Online]. Available: https://maikel.pro/blog/whatsspy-public-support-ending-today

APPENDIX

A. Differences in the Number Spaces

Interestingly, the amount of registrable mobile phone numbers greatly differs between countries and is not necessarily proportional to the country's population. In Tab. VI we list a selection of countries with their amount of registrable mobile phone numbers (i.e., filtered by `libphonenumber` and our WhatsApp registration API check) and set it in relation to the population. The ratio between the number space and the population indicates whether the amount of resources spent enumerating the entire number space can yield a satisfactory amount of matches. For example, while the US and Germany have roughly the same amount of registrable mobile phone numbers, one would expect to find many more active phone numbers in the US due to the much larger population.

Our results show that small as well as less developed countries often have a limited number space and therefore can be crawled with very little effort. On the other hand, we observe some outliers: Austria, for example, has such a large number space that for every citizen there are more than 10,000 registrable mobile phone numbers available. While such a ratio seems to make crawling infeasible, one can still exploit the fact that the phone numbers are typically not uniformly distributed, but given away in blocks. Hence, one could follow the strategy to first randomly check a few numbers for each possible prefix and then focus on the most fruitful prefixes to still cover a good portion of the population.

B. Reduction Function for Our Optimized Rainbow Tables

The reduction function for our optimized rainbow tables converts a hash value back to a valid (mobile) phone number (cf. § III-C). A trivial reduction function could be defined by taking the first 64 bit h_{64} of hash h and calculating the modulus with the total amount of phone numbers N, giving us the index of the phone number in the table of all numbers.

Country	# Numbers (in million)	# Numbers / Population
Cuba	0.3	0.03
Moldova	6.3	1.9
Australia	48.6	1.9
Canada	76.0	2.0
Japan	127.9	1.0
Russia	327.0	2.2
United States	505.7	1.5
Germany	538.3	6.5
China	4,496.0	3.2
Austria	93,658.7	10,573.4

Table VI: Comparison of countries with regard to their amount and density of registrable mobile phone numbers.

However, the modulo operation introduces non-uniformity in the output of the reduction function: The lower parts of the number space are more likely to be chosen if N is not a divisor of 2^{64}. We therefore introduce an offset into the calculation that varies for every chain index i_C. By choosing the offset as the division of N by the chain length l_C, each chunk of the phone number space is more likely for one chain index, producing a uniform distribution overall.

Another issue is collisions in the reduction function: As soon as two different hashes produce the same phone number, then all successive elements of the chain would be duplicate entries. To prevent this, we vary our reduction function with every chain index as well as with every table[15].

Therefore, we define our reduction function as follows:

$$i = \left(h_{64} + \left(\left\lfloor \frac{N}{l_C} \right\rfloor + 65{,}536 \times i_T \right) \times i_C \right) \mod N,$$

where the different parameters are explained in Tab. VII.

Parameter	Explanation
h_{64}	First 64 bit of hash h encoded as an unsigned integer
N	Number of possible plaintexts
l_C	Length of the chains
i_T	Index of the currently generated table
i_C	Current index in the chain

Table VII: Parameters for our reduction function.

The "magic" number 65,536 for the table offset was kept from the original RainbowCrack implementation [35], since it produces reasonable results.

C. A Note on Per-Character Alphabets

Rainbow table implementations like CryptoHaze [7] allow to specify individual alphabets for each character position of the input domain. While this might appear to be a solution to optimize rainbow tables for (mobile) phone numbers, the unique structure of phone numbers limits the usefulness of this technique: the possibilities for each digit in a phone number are strongly dependent on the preceding digits.

German mobile phone numbers can be used to illustrate this point, since they always start with the digits `+491`. If one were to limit the third character of the input domain only to `1`, the input space of the rainbow table could be reduced by a factor of 10x. However, phone numbers from all other countries where the third digit is not `1` would be missing from the resulting table. While it would be possible to generate rainbow tables with different alphabets for each country or even mobile prefix, this would require considerable effort and performance overhead, and ultimately closely resemble our approach (at least conceptually).

[15] Rainbow tables are usually split into several files due to their large size.

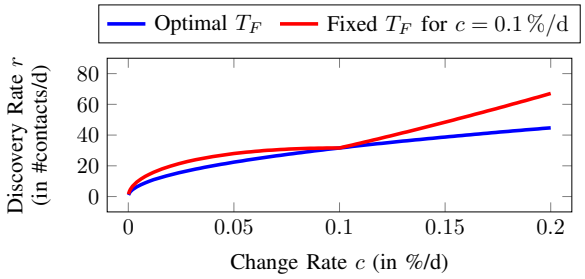


Figure 4: Minimal discovery rate for different change rates c with optimal choices for T_F (for Signal's parameters) compared to the discovery rate for fixed T_F when estimating $c = 0.1\,\%/\mathrm{d}$.

D. Further Optimizations for Hash Reversal Methods

Given that the mobile number prefixes in our rainbow table construction can be chosen freely (cf. § III-C), it is possible to construct tables for arbitrary subsets of all phone numbers, such as for one or multiple countries, or limited to a certain length or type. As a result, storage space requirements and lookup time can further be reduced for specific applications.

Splitting the tables into countries also allows probabilistic searches based on some known or learned user distribution (e.g., if most users of a service are from the US, the rainbow table containing US phone numbers will be searched first), or other available meta data (e.g., IP addresses, or previously cracked numbers from the same address book).

Our experiments also reveal non-uniform distributions of phone numbers within single countries (cf. § IV), which could be used to further speed up the reversal process.

Hybrid constructions of hash databases (cf. § III-A) and brute-force (cf. § III-B) can outperform each individual method, since small batches of numbers, for which hashcat has significant overhead, can be handled efficiently by the hash database.

E. Optimal Parameters for Incremental Contact Discovery

Given that the popularity of mobile messengers fluctuates over time, the change rate c of the server database is not a fixed value but varies continuously. This results in different optimal choices for the time T_F. The inevitable non-optimal values for T_F between adjustments result in higher discovery rates than the possible minimum: If c is higher than expected, more users can be found by observing S_D. If c is lower than expected, the rate limits for S_F are too generous.

The relative error between the minimal and the actual discovery rate can be calculated as $e = 0.5 \cdot |1 - c/c_{est}|$, where c is the actual change rate and c_{est} is the estimated one used for setting T_F. Thus, if the real change rate is underestimated by a factor of 2x, the discovery rate will be 50 % higher than intended. For the parameters used by Signal (cf. § V), Fig. 4 shows how the discovery rate behaves compared to the minimal one when a constant change rate of $c = 0.1\,\%/\mathrm{d}$ is assumed. Obviously, underestimating the change rate is more problematic than overestimating it. In a production environment it therefore may be beneficial to set c slightly higher than the expected value to deal with fluctuations. An implementation with dynamic sets, as outlined in § V, could be an option for platforms where the change rate fluctuates more strongly and frequent adjustments of T_F are required.

F. Supplemental Mitigation Techniques

A number of additional strategies could potentially supplement the mitigations discussed in § VI, such as CAPTCHAs for users with unusual contact discovery patterns, honeypot numbers [40] to detect enumeration attacks, modeling user behavior for anomaly detection, or the increase of the phone number space by telecommunications providers. Yet these approaches are either impractical (larger phone number space), can result in a high number of false positives (honeypot numbers, behavioral analysis), require the processing of user data (behavioral analysis), or decrease usability (CAPTCHAs). For the sake of completeness, we nevertheless discuss each of these techniques shortly in the following.

Increased Phone Number Entropy. In § A, we observed that some countries have a much larger number space than others, which makes crawling these countries much more difficult. Telecommunication companies of vulnerable countries could therefore agree to maintain larger number blocks to increase the search space for attackers. However, it is important that the numbers are randomly distributed such that there are no clusters that can be efficiently crawled once detected by an attacker. While this approach also makes hash reversal more difficult, we demonstrated in § III that it is feasible even for countries with a large number space (e.g., Austria).

CAPTCHAs. In countless web applications, CAPTCHAs are in place to prevent automated API abuse. Even though there are ways to circumvent CAPTCHAs [52], [86], they still can significantly slow down an attack or at least increase the cost of abuse. Therefore, we suggest to also use CAPTCHAs in mobile messaging applications to differentiate legitimate users with unusual contact discovery patterns from abusers.

Modeling User Behavior. Service providers could use heuristics to detect abnormal user behavior that indicates a crawling attempt. Such heuristics could include an unusually large amount of contacts in the address book, exceptionally many syncing requests, and constantly changing contacts. However, using such heuristics to automatically ban accounts is error-prone. This kind of detection can also be circumvented by more sophisticated attackers that adapt their behavior to evade detection.

Honeypot Numbers. Rate limits can be bypassed by sophisticated attackers, e.g., by crawling with a low rate. As was also suggested in [40], service providers could use honeypots for detection of such attackers: They could acquire several phone numbers themselves and detect if any of these numbers are matched during contact discovery. A positive match would indicate either a false positive (e.g., a typo when storing a contact) or an attempt of crawling. Due to the potential of false positives, it would be more reasonable to closely monitor the activity of such accounts rather than blocking them instantly.

Educating Users. Users might not be aware of the fact that their public information is indeed easily accessible to third parties that perform data scraping. Messaging applications therefore could show reminders about this fact whenever users are in the process of sharing personal information publicly, e.g., when uploading a public profile picture.

Additionally, on-device machine learning techniques could be applied to automatically educate users about the sensitivity of shared content, e.g., when extended nudity or children are detected in uploaded profile pictures.

B Mobile Private Contact Discovery at Scale (USENIX Security'19)

[KRS+19] D. KALES, C. RECHBERGER, T. SCHNEIDER, M. SENKER, C. WEINERT. **"Mobile Private Contact Discovery at Scale"**. In: *28. USENIX Security Symposium (USENIX Security'19)*. Website: https://contact-discovery.github.io. Full version: https://ia.cr/2019/517. USENIX Association, 2019, pp. 1447–1464. CORE Rank A*. Appendix B.

https://www.usenix.org/system/files/sec19-kales.pdf

Mobile Private Contact Discovery at Scale

Daniel Kales and Christian Rechberger, *Graz University of Technology;*
Thomas Schneider, Matthias Senker, and Christian Weinert, *TU Darmstadt*

https://www.usenix.org/conference/usenixsecurity19/presentation/kales

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

Mobile Private Contact Discovery at Scale

Daniel Kales
Graz University of Technology

Christian Rechberger
Graz University of Technology

Thomas Schneider
TU Darmstadt

Matthias Senker
TU Darmstadt

Christian Weinert
TU Darmstadt

Abstract

Mobile messengers like WhatsApp perform contact discovery by uploading the user's entire address book to the service provider. This allows the service provider to determine which of the user's contacts are registered to the messaging service. However, such a procedure poses significant privacy risks and legal challenges. As we find, even messengers with privacy in mind currently do not deploy proper mechanisms to perform contact discovery privately.

The most promising approaches addressing this problem revolve around private set intersection (PSI) protocols. Unfortunately, even in a weak security model where clients are assumed to follow the protocol honestly, previous protocols and implementations turned out to be far from practical when used at scale. This is due to their high computation and/or communication complexity as well as lacking optimization for mobile devices. In our work, we remove most obstacles for large-scale global deployment by significantly improving two promising protocols by Kiss et al. (PoPETS'17) while also allowing for malicious clients.

Concretely, we present novel precomputation techniques for correlated oblivious transfers (reducing the online communication by factor 2x), Cuckoo filter compression (with a compression ratio of $\approx 70\,\%$), as well as 4.3x smaller Cuckoo filter updates. In a protocol performing oblivious PRF evaluations via garbled circuits, we replace AES as the evaluated PRF with a variant of LowMC (Albrecht et al., EUROCRYPT'15) for which we determine optimal parameters, thereby reducing the communication by factor 8.2x. Furthermore, we implement both protocols with security against malicious clients in C/C++ and utilize the ARM Cryptography Extensions available in most recent smartphones. Compared to previous smartphone implementations, this yields a performance improvement of factor 1,000x for circuit evaluations. The online phase of our fastest protocol takes only 2.92 s measured on a real WiFi connection (6.53 s on LTE) to check 1,024 client contacts against a large-scale database with 2^{28} entries. As a proof-of-concept, we integrate our protocols in the client application of the open-source messenger Signal.

1 Introduction

After installation, mobile messaging applications first perform a so-called *contact discovery*. This allows new users to automatically connect with all other users of the messaging service whose phone numbers are stored in their address book. There exist various ways to perform contact discovery. For example, WhatsApp simply uploads the user's entire address book on a regular basis to match contacts [1].

However, revealing all personal contacts to a service provider poses significant privacy risks: from the social graph of users a variety of personal information can be inferred and journalists, for example, may need to cover the identity of some of their informants to protect whistleblowers from potential consequences. When installing a mobile messaging application, users also jeopardize the privacy of people who are not even connected to the particular service by transmitting their contact information without consent. An illustrative example of a severe breach of privacy can be seen in the case of WhatsApp, which was acquired by Facebook in 2014 and shared its database with the parent company: Facebook users received friend recommendations of strangers who happened to see the same psychiatrists [33].

Unfortunately, applying simple protection mechanisms like hashing the phone numbers of contacts locally before the upload to the service provider is not helpful since these hashes are vulnerable to brute-force and dictionary attacks due to the relatively small range of possible pre-images. Furthermore, the service provider can still tell whether two users share a contact even a long time after running the discovery routine by storing the received hash values. Custom wrappers[1] for messaging applications can somewhat circumvent these problems by allowing users to manually select contacts to expose to the messaging application. However, this approach only protects the contacts of users actually using such custom wrappers. Furthermore, manually selecting the contacts to match is a usability problem.

[1] e.g., https://www.backes-srt.com/en/solutions-2/whatsbox

One possible solution to this dilemma is to apply a particular form of secure two-party computation. In general, secure two-party computation allows parties P_1 and P_2 to jointly compute a publicly known function f on their respective inputs X_1 and X_2 s.t. the parties learn no information from the protocol execution but the result. The research area of *private set intersection* (PSI) focuses on optimized protocols for the case where X_1 and X_2 are sets of elements, and f is the intersection function. PSI has been studied in great depth in the past years, yielding very efficient protocols (e.g., [41, 51]) based on oblivious transfer extensions (OTe, cf. [4, 36, 39]). However, while these protocols are very efficient in many scenarios, they turn out to be impractical for use-cases like private contact discovery on mobile devices, where the input set of the service provider is much larger (sometimes by a factor of a few million) than the input set of the user. This is because the online phase of these protocols (which depends on the actual inputs) has a computation and communication complexity that is linear in the size of the larger set.

Therefore, other PSI protocols for the case of *unbalanced* set sizes were developed (e.g., [19, 21, 40, 59]). However, only [40] actually provides an implementation on real mobile smartphone clients. The experiments performed by the authors of [40] show a rather large discrepancy between protocol execution on x86-based PC hardware and Android smartphones where performance-critical cryptographic operations are implemented in Java. In fact, their performance results do not encourage real-world deployment. For example, their fastest protocol that can easily be made secure against malicious clients requires more than 52 s on a smartphone with WiFi connection to check a single client contact against a database with only 2^{20} entries.

The developers of Signal, a mobile messaging service similar to WhatsApp but with focus on privacy, considered the use of PSI protocols for contact discovery. However, they refrained from actually implementing PSI since the academic research in PSI and the related private information retrieval (PIR) protocols "is quite a disappointment" [44]. Instead, they presented a technology preview that protects the contact discovery task on the server side with Intel Software Guard Extensions (SGX), a trusted execution environment that can be attested by remote users [45]. In theory, this yields a secure contact discovery service with negligible performance overhead compared to plain computation. However, Intel SGX is a proprietary engineering-driven solution with no cryptographic security guarantees and vulnerable to severe attacks, e.g., the recent Foreshadow attack [16] managed to reliably extract confidential data from enclaves. Moreover, some fixes for hardware security designs such as Intel SGX require hardware changes that can take years to enter the market and result in repeated acquisition costs. In contrast, fixes for flawed implementations of provably secure cryptographic protocols can be deployed quickly via software updates.

Thus, we revisit state-of-the-art unbalanced PSI protocols which provide cryptographic security and show that using new optimizations and native implementations they turn out to be practical on modern smartphones. Furthermore, we achieve security against malicious clients: since every user could run a manipulated version of the messaging application, deviations from the protocol may lead to revealing information about the server's database. On the other hand, we assume that the server behaves semi-honestly, i.e., it follows the protocol but tries to learn as much information as possible. This is a reasonable assumption since there are legal requirements and financial incentives to behave correctly: once misconduct gets known publicly, users will abandon the misbehaving service and switch to a more trustworthy alternative.

1.1 Our Contributions

As a motivation, we investigate how contact discovery is handled in widely used mobile messaging applications. For this, we conduct a survey where we analyze privacy policies, source code, and network traffic. Our results show that in practice none of these applications protect the users' privacy during contact discovery.

We optimize two protocols for unbalanced PSI that can easily be made secure against malicious clients and are suitable for private contact discovery: one that uses oblivious evaluations of the Naor-Reingold PRF (NR-PSI, cf. [31, 40, 47]) and one that uses Yao's garbled circuits (GC-PSI, cf. [40, 52, 56]) to run oblivious AES evaluations. For both protocols we apply new forms of correlated random OT precomputation (reducing the online communication by factor 2x, which is of independent interest) and introduce a method for Cuckoo filter compression (with a compression ratio of $\approx 70\,\%$ and negligible computational overhead) as well as 4.3x smaller Cuckoo filter updates to reduce the required network communication. Moreover, we improve the GC-PSI protocol by instantiating the PRF with LowMC [2], a cipher specifically designed for efficient evaluation in secure protocols, instead of the default choice AES. While this was already proposed in [40], we find optimal parameter sets for LowMC and provide implementations. Compared to AES, we thereby reduce the communication by factor 8.2x.

We provide C/C++ implementations for both protocols with security against malicious clients that make use of the Cryptography Extensions (CE) in the ARMv8 architecture available in most recent smartphones for hardware-accelerated execution. Thereby, we improve the runtime of the online phase of the GC-PSI protocol by more than a factor of 1,000x compared to the previous work of [40] that only implements security against semi-honest clients. We overcome further shortcomings of previous works w.r.t security and scalability by evaluating the implementations using recommended security parameters, reasonable false positive probabilities, and considering large-scale set sizes on the server side.

Our fastest protocol takes only 2.92 s measured on a real WiFi connection (6.53 s on LTE) and 6.07 MiB of communication in the online phase to check 1,024 client contacts against a database with 2^{28} entries (more than the number of monthly active users for popular messengers like Telegram [61]). For the setup phase it is required to transfer a compressed Cuckoo filter once whose size is linear in the number of the database entries ($\approx$ 1 GiB for 2^{28} entries); since the filter is identical for all clients, service providers can handle the resulting traffic efficiently via CDNs. To remain practical for even larger set sizes (the market leader WhatsApp currently has more than 1.6 billion users [61]), we suggest multiple extensions, e.g., combining our protocols with multi-server PIR s.t. the overall client-server communication complexity becomes logarithmic in the size of the server database.

As a proof-of-concept, we integrate both of our protocols in the Signal Android client, thereby positioning our secure cryptographic approach as a practical alternative to vulnerable trusted execution environments like Intel SGX.

1.2 Motivating Survey

To determine how contact discovery is currently being done in practice, we conducted a survey on a comprehensive selection of mobile messengers that are "secure" in the sense that they offer end-to-end encryption. Each application was analyzed by evaluating the mandatory privacy policy, which is supposed to state exactly which data the application transmits to its server and how the server processes and stores that data. Unfortunately, these policies are not always precise enough to determine the employed contact discovery method. In these cases, we inspected the source code (if publicly available) or the network communication by means of the man-in-the-middle proxy *mitmproxy*[2]. We circumvented certificate pinning by using the *Xposed*[3] framework together with the *JustTrustMe*[4] plugin that can disable certificate checking routines in several commonly used security libraries.

Our results are summarized in Tab. 1. All surveyed messengers upload contact information (at least the contact's phone number) either in the clear or in hashed form. While this form of contact discovery is very efficient (requiring only a few bytes of communication per element), it threatens the privacy of users directly or indirectly via brute-force or dictionary attacks. Furthermore, even if the server cannot determine the actual contact data, it can still tell whether two users share a contact by comparing uploaded hash values.

This can be somewhat mitigated by using salted hashing s.t. the hashes received by the server are different whenever a client triggers contact discovery. However, only one of the surveyed messengers employs this approach as it requires to hash the entire server database for each fresh salt received by a client. Furthermore, brute-force attacks are still feasible.

Messenger	Hashed	Salted	Analysis Technique
Confide*	✓	✗	Privacy policy
Dust*	✗	✗	Network traffic
Eleet*	✗	✗	Privacy policy
G DATA Secure Chat	✓	✗	Network traffic
Signal (legacy)	✓	✗	Source code
SIMSme	✓	✓	Network traffic
Telegram	✗	✗	Privacy policy
Threema	✓	✗	Privacy policy
Viber	✗	✗	Privacy policy
WhatsApp	✗	✗	Privacy policy
Wickr Me	✓	✗	Privacy policy
Wire	✓	✗	Privacy policy

Table 1: Results of our contact discovery survey on secure mobile messengers. All applications upload contact information either in the clear or hashed (with salt). Messengers marked with * denote that contact discovery is optional.

2 Related Work

In this section, we discuss existing unbalanced PSI protocols and other works that focus on PSI in the smartphone setting.

Unbalanced PSI. Kiss et al. [40] discuss multiple unbalanced PSI protocols with precomputation (cf. §3.5) and security against semi-honest adversaries. Their NR-PSI and GC-PSI protocols (based on [31] and [52], respectively) are the foundation of our work. We augment these protocols with new OT precomputation techniques, efficient Cuckoo filters [27, 59], a specialized cipher [2] for the GC-PSI protocol, and security against malicious clients. The authors of [40] also evaluate their protocols on smartphones, but based on less efficient Java implementations. In our work, we present C/C++ implementations that make use of the hardware-accelerated cryptography available in most recent smartphones.

Resende and de Freitas Aranha [59] use techniques similar to [40], but replace Bloom filters [12] with the more efficient and versatile Cuckoo filters [27] to efficiently represent the encrypted server database (cf. §3.4) in a Diffie-Hellman style PSI protocol [7] with security against semi-honest adversaries. In our work, we optimize communication by proposing methods for Cuckoo filter compression and updates, and perform evaluations with reasonable parameters: while in [59] the authors settle with an error probability of $\approx 2^{-13}$, which results, on average, in one false positive when 10 clients match 2^{10} contacts each, we propose realistic Cuckoo filter parameters for error probabilities $\approx 2^{-29}$ and $\approx 2^{-39}$.

Demmler et al. [21] present a different approach assuming multiple non-colluding servers. Their idea is to first perform a variant of private information retrieval (PIR) to reduce the

[2] https://mitmproxy.org
[3] https://repo.xposed.info
[4] https://github.com/Fuzion24/JustTrustMe

server's input set and then perform a traditional PSI protocol on the reduced sets. While this approach is very performant, the requirement of non-colluding servers presents challenges for the data-owners: they not only need to guarantee that these servers do not collude, but also need to ensure that their client data is not leaked to other parties. This leads to the difficult situation where the server party needs to trust a second server but simultaneously is assumed to not collude with it. However, even if servers are malicious and/or collude, they cannot learn more about client inputs than in currently deployed naive hashing-based contact discovery methods.

Chen et al. [19] give a PSI protocol based on fully homomorphic encryption. The authors present multiple optimizations that make the protocol practically viable. Their work was improved and extended to the special use case of *labeled* PSI [18], where for intersecting items an associated label is transferred and security is not only guaranteed in case of malicious clients but also malicious servers (with some controlled leakage). The advantage of the protocols of [18, 19] is that their communication complexity is sublinear instead of linear in the size of the server set. However, this comes at the cost of repeated high computational overhead, whereas the online phase of our protocols is very efficient and requires no cryptographic operations on the server side.

Mobile PSI. Huang et al. [34] provided first performance results for secure computation on smartphones with security against semi-honest adversaries. They implemented a circuit-based PSI protocol on Android. Their implementation managed to evaluate ≈ 100 AND gates per second, taking about 10 min to intersect two sets of 256 items each.

Asokan et al. [6] implemented an RSA-based PSI protocol with security against semi-honest adversaries on smartphones for secure mobile resource sharing.

Carter et al. [17] presented a maliciously secure system for secure outsourced garbled circuit evaluation on mobile devices. Subsequently, Mood et al. [46] showed how to further optimize outsourced evaluation. They also point out how their framework can be used to implement a secure friend finder.

"PROUD" [49] is a decentralized approach for private contact discovery based on the DNS system. It enables users to privately discover the current network addresses of friends, which differs from the scenario of a centralized messaging service we consider. Moreover, friendship bootstrapping requires an out-of-band communication channel between users.

Compared to these works, we optimize protocols for unbalanced PSI with a central service provider and provide native implementations for maximum performance on smartphones.

3 Background

In the following, we introduce cryptographic building blocks that are required for the remainder of this work.

3.1 Oblivious Transfer (Extensions)

Oblivious transfer (OT) [57] is a cryptographic protocol that in its most basic form allows a sender P_1 to obliviously transfer one out of two messages (m_0, m_1) to a receiver P_2 based on a selection bit b chosen by P_2 s.t. P_1 learns nothing about b and P_2 learns only m_b but nothing about m_{1-b}.

It was shown in [35] that performing OTs always requires some form of public key cryptography. However, with OT extension (OTe) protocols [9, 36], a small number (e.g., 128) of "base OTs" can be extended to a large number of OTs using only efficient symmetric cryptographic operations.

There exist flavors of OTe with reduced communication complexity [5]: In random OT (R-OT), neither party inputs any values, but the inputs of sender and receiver are randomly chosen by the protocol. In correlated OT (C-OT), m_0 is chosen at random, whereas m_1 is computed as a function f of m_0: $m_1 = f(m_0)$, where f is privately known to P_1 only.

It is possible to precompute OTs s.t. all computationally expensive operations are performed via R-OTs in advance [8]. Later, the random values obtained via R-OTs are used to mask the actual inputs, requiring only cheap XOR operations in the style of one-time-pad encryption.

3.2 Garbled Circuits

Yao's garbled circuits (GC) [62] is one of the most prominent techniques for secure two-party computation. (In the following the two parties are called *garbler* and *evaluator*.) The idea is to represent the function that is evaluated as a Boolean circuit and to replace each logical two-input gate by a *garbled gate*. Each wire of the garbled gate is given two random wire labels, representing 0 and 1. To garble a gate, the garbler uses all four combinations of the gate's two input wire labels to encrypt the corresponding output wire label, based on the truth table of the original gate, and sends the resulting ciphertexts, the so-called *garbled table*, to the evaluator. The evaluator can then use the two input wire labels it possesses to decrypt one of the four ciphertexts and receive the output wire label, which is then used as input for subsequent gates.

We now describe how the evaluator obtains the wire labels corresponding to the inputs of the two parties: Since the garbler knows all wire labels, it can send the wire labels corresponding to its input bits to the evaluator. However, to ensure input privacy for the evaluator, the wire labels corresponding to the evalutor's input bits are retrieved via OTs. The garbler also sends information that allows the evaluator to decode the final output wire labels to 0 or 1.

Several optimizations for Yao's original scheme have been presented s.t. today it is most efficient to combine the following techniques: Point-and-Permute [10], Free-XOR [42], fixed-key AES garbling [11], and Half-Gates [63].

3.3 OPRF Evaluation

An oblivious pseudorandom function (OPRF) is a protocol between two parties: sender P_1 holding key k and receiver P_2 holding input x. After the invocation of the protocol, P_2 learns the output $f_k(x)$ of a keyed pseudorandom function (PRF) f. Additionally, it is guaranteed that P_1 does not learn anything about x and P_2 does not learn anything about k.

OPRF evaluations can be used to build PSI protocols as proposed in [28, 30, 40, 52]: The server samples a key k uniformly at random, evaluates the PRF $f_k(x_i)$ on each of its items $x_i \in X$, and sends the results to the client. Server and client now engage in the OPRF protocol, where the server inputs key k and the client inputs elements $y_j \in Y$. After this step, the client obtains $f_k(y_j)$ for each item $y_j \in Y$ and can perform a plain intersection between the items $f_k(x_i)$ and $f_k(y_j)$. The client then outputs the elements y_j corresponding to the values in the intersection.

In this work, we instantiate the PRF either using the Naor-Reingold PRF [47] (NR-PSI) or a garbled circuit-based evaluation of a block cipher (GC-PSI). In [37], the authors describe an alternative algebraic OPRF construction based on a PRF by Dodis-Yampolskiy [25]. However, due to the use of Paillier encryption, this construction is likely slower than the Naor-Reingold PRF and their follow-up work [38], the basis for [59] (cf. §6.2). Moreover, it requires a common reference string in the form of an RSA modulus with unknown factorization.

3.4 Cuckoo Filters

Cuckoo filters [27] are an alternative to the more popular Bloom filters [12]. Like Bloom filters, they are a data structure for compact set representation that allows for fast membership testing with controllable *false positive probability* (FPP). Cuckoo filters employ a hashing technique similar to Cuckoo hashing [48], which has been used in the past as a building block in PSI protocols (e.g., [41, 51, 53–56]).

Resende and de Freitas Aranha [59] first used Cuckoo filters in a PSI protocol. This is due to several advantages over Bloom filters when representing the server's database, namely they (i) support inserting and deleting items subsequently, whereas standard Bloom filters only support inserting items, and variants that do support deletion such as counting Bloom filters have much higher storage costs; (ii) have better lookup performance; and (iii) use less space in many scenarios while having the same false positive probability.

Cuckoo filters consist of a table of buckets with fixed bucket size b. Inside the buckets, so-called tags are stored. Tags are small bitstrings obtained by hashing items. More precisely, to represent an item x in a Cuckoo filter, we first calculate its tag $t_x = H_t(x)$, where H_t is a hash function with output bitlength v. This tag is stored in one out of two possible buckets. The position of the first possible bucket is calculated as $p_1 = H(x)$, where H is another hash function that maps the input to a position in the table of buckets. In case this bucket is already full, the tag is stored in the second possible bucket at position $p_2 = p_1 \oplus H(t_x)$. Note that it is always possible to determine the other candidate bucket p_j just from knowing its tag t_x and the current position p_i: $p_j = p_i \oplus H(t_x)$. If both buckets are full, one tag in one of the buckets is chosen at random, removed from that bucket, and moved to its other possible bucket. This procedure is repeated recursively until no more relocations are necessary.

To check whether an item is contained in the Cuckoo filter, one computes its tag and both possible bucket locations and compares the tags stored there for equality. For deleting the item, the matching tag is removed from the filter.

Due to hash collisions, two items may produce equal tags. As a consequence, lookups can lead to false positives. The false positive probability ε_{max} is mainly dependent on the tagsize v and also slightly on the bucket size b since larger buckets result in more possible collisions within each bucket.

3.5 Unbalanced PSI with Precomputation

For private contact discovery, the following properties are desired: (i) the server performs the computationally expensive tasks; (ii) all computationally expensive and communication intensive tasks are performed only once; and (iii) the actual intersection computation is very fast and also allows for efficient updates. Therefore, [40] suggest to use PSI protocols with precomputation, where most time consuming tasks are performed ahead of the actual intersection.

Our PSI protocols for unbalanced set sizes share a common structure. Following the precomputation approach of [40], they are divided into the following four phases: (i) The *base phase* is completely independent of any input data and consists, e.g., of OT precomputation. Its complexity is linear in the maximum number of contacts a client expects to match in future protocol executions before the base phase is re-run. (ii) The complexity of the *setup phase* is linear in the size of the large set held by the server. It involves encrypting all elements in the server database via PRF evaluations as described in §3.3 and inserting them into a Cuckoo filter for compact representation, which is transferred to the client. (iii) During the *online phase* client and server jointly perform OPRF evaluations on all elements of the client. The client then looks up all received encryptions in the Cuckoo filter to determine the intersection. Thus, the complexity of the online phase is only linear in the size of the small client set. (iv) Changes in the server database trigger the *update phase*, where the Cuckoo filter on the client side is updated by sending a small delta for each inserted or deleted database entry.

4 Optimizing OPRF-based PSI Protocols

We propose more efficient database representations and PRFs, give the full descriptions for our optimized NR- and GC-

PSI protocols, enable security against malicious clients, and suggest multiple extensions to further increase practicality.

4.1 More Efficient Database Representations

Realistic Cuckoo Filter Parameters. Resende and de Freitas Aranha [59] propose using Cuckoo filters as an extension to the DH-based PSI protocol of [7] and they perform experiments to find optimal Cuckoo filter parameters based on the number of server items and the desired error probability. While their findings are directly applicable to our use case, they set very aggressive Cuckoo filter parameters (tagsize $v = 16$, bucket size $b = 3$) and settle for a maximum *false positive probability* (FPP) of $\varepsilon_{max} \approx 2^{-13}$. We find this FPP not practical since it implies that about one in 10 clients performing PSI for 2^{10} elements receives a false positive.

Instead, we propose to use tagsize $v = 32$ to reach a FPP of $\varepsilon_{max} \approx 2^{-29}$ or tagsize $v = 42$ to reach a FPP of $\varepsilon_{max} \approx 2^{-39}$ while still maintaining a bucket size of $b = 3$. For our experiments, we choose the parameter set $v = 32, b = 3$, and choose the size of the Cuckoo filter to have a load factor of $\approx 66\%$, leading to a Cuckoo filter size of 6 MiB per 2^{20} items.

Novel Cuckoo Filter Compression. The size of Cuckoo filters can be reduced by applying a simple but effective compression technique that to the best of our knowledge was not considered before: For each entry of a Cuckoo filter, an additional bit is transmitted that indicates whether this entry is empty or holds a tag. The entry itself is only transmitted if it is not empty. This way, the filter is represented as a bit map and a list of tags. For a Cuckoo filter storing n items with tagsize v, bucket size b, and load factor l, this reduces the size from $\frac{n}{l} \cdot v$ bits to $\frac{n}{l} + n \cdot v$ bits. In the example above, the size of the Cuckoo filter is reduced from 6 MiB to 4.19 MiB, i.e., by $\approx 30\%$. An advanced version of the compression technique presented above encodes the number of tags (0 to b) in each bucket with $\log_2(b+1)$ bits instead of sending b bits per bucket. This is possible since the actual position of each tag within a bucket is not important.

This compression technique is especially useful for very sparse Cuckoo filters, which appear in use cases where the set of items is expected to grow fast (e.g., during the release phase of a new messaging application). For example, if only 10 % of a Cuckoo filter storing a maximum of 2^{20} items is occupied, it can be compressed by a factor of 8.3x.

In concurrent and independent work, Breslow and Jayasena [15] proposed *Morton filters*, which combine these compression techniques with cache-optimized layouts and further optimizations. Morton filters provide higher insertion, lookup, and deletion throughput than traditional Cuckoo filters, while usually having equal or slightly lower storage costs. We leave the evaluation and usage of Morton filters in our protocols for future work.

Better Cuckoo Filter Updates. In [59], when performing an update after new elements are inserted into or deleted from the server's set, each encrypted element to be updated is sent to the client where it is inserted into the existing Cuckoo filter. However, for Cuckoo filters, all information required to insert a new item is its tag and the index of one of its candidate buckets. From this information, it is possible to calculate the second candidate bucket in case relocations are necessary. The same information is also sufficient to delete an item. For example, the bucket index in a Cuckoo filter storing $n = 2^{28}$ items with bucket size $b = 3$ and load factor $\approx$ 66 % can be represented with 27 bits. This results in sending 59 bits per updated element for tagsize $v = 32$. In comparison, in [59] an encrypted element is represented by one point on the GLS-254 binary elliptic curve, which results in 256 bits of communication when using point compression with two trace bits, which needs 4.3x more communication than our approach.

4.2 More Efficient PRF for GC-PSI

During the online phase of the GC-PSI protocol, both parties interactively evaluate an OPRF on the client's items using garbled circuits. For each of the client's items, the server prepares a garbled circuit $\widetilde{PRF}_k$ that evaluates the chosen PRF under the server's key k. The choice of this PRF has a significant impact on both the runtime and the communication complexity of the overall protocol. Several improvements for Yao's GC protocol [62] have appeared in recent years that changed the desired properties of the functions to be evaluated. Most notably is the Free-XOR [42] optimization, which allows XOR gates to be evaluated securely "for free", meaning all necessary operations can be performed locally without any communication between the parties. This optimization has lead to research in the area of ciphers with a low number of AND and instead many free XOR gates.

In previous GC-PSI implementations, the choice of the PRF was AES-128. Using the optimized S-Box implementation of [13], an AES-128 circuit (without key schedule) has 5,120 AND gates [32], serving as a baseline for comparison.

In this section, we focus on variants of LowMC [2], a highly parameterizable block cipher designed for use cases in multiparty computation (MPC) and fully-homomorphic encryption (FHE). [40] mentioned the possibility of using LowMC instead of AES for GC-PSI. We look at several instantiations of LowMC and present optimized parameter sets specifically for the use case of GC-PSI and mobile contact discovery. In the following, we give a short description of LowMC and highlight the different parameter choices.

LowMC [2] is a block cipher where block size n, key size k, number of S-Boxes per substitution layer m, and allowed data complexity d can be chosen freely up to some sanity constraints. The required number of rounds r to reach the security claims is then derived from these parameters.

Data Complexity. The data complexity of a cipher is the number of plaintext-ciphertext pairs allowed to be released before the security claims no longer hold. In the GC-PSI protocol, we can exactly control the maximum number of published plaintext-ciphertext pairs by limiting the number of client queries, and therefore can reduce the number of LowMC rounds required for security. We set the allowed data complexity to be $d = 2^{64}$, allowing for 2^{20} contact discoveries of 2^{10} items for each of the 2^{28} clients, while still being below the security margin by a factor of over 100x. For smaller-scale applications, we also give a parameter set for 2^{32} total data complexity, which suffices to run 2^{20} queries of 2^{10} items each. While we could also use this parameter set for larger-scale applications, the system needs to be re-keyed after the data complexity has been reached.

Key Schedule. In many MPC applications using OPRF evaluations, one party knows the entire secret key and can, therefore, perform any key-scheduling algorithm (e.g., for AES or LowMC) offline. The circuit is then modified to take the expanded key as an input. In many cases, this can be a performance improvement since the key-schedule algorithm does not have to be computed using the MPC protocol. However, when performing OPRF evaluations using garbled circuits, the party holding the secret key needs to send wire labels for each input bit, increasing the communication. While for AES-128, only 11x more wire labels need to be transferred for the expanded key, some instantiations of LowMC require several hundreds of rounds. Sending labels for the expanded key essentially removes the advantage of the lower AND count that comes with such a large number of rounds. However, we observe that in the GC-PSI protocol the OPRF evaluation is always performed with the same key. Thus, we can bundle all of the client's circuits together into one large circuit and evaluate the key-schedule only once. This means that we only need to send the wire labels corresponding to the non-expanded key once, and therefore save ≈ 2 KiB for each subsequent client item when using a 128-bit key. It is also possible to only evaluate parts of the garbled circuit if the number of client items is lower than the number of precomputed circuits.

LowMC Instances. For use in our GC-PSI protocol, we highlight several LowMC instances, exploring different parameter choices. In Tab. 2, we give the parameters and compare the number of AND gates to AES-128. The number of rounds is calculated according to the LowMCv3 round formula[5], which was updated by the LowMC team to take new cryptanalysis of LowMC (cf. [23, 24, 58]) into consideration. We can observe some interesting properties: LowMC instances (1) and (2) require the same number of rounds to be secure, but instance (1) has the maximum number of possible S-Boxes, while (2) does not. Since instance (2) provides the same security as (1) while requiring fewer S-Boxes, and therefore a lower amount of AND gates, it should always be preferred. LowMC instance (3) has the smallest possible S-Box layer with only one S-Box per round and also the lowest number of AND gates. While its 208 rounds can be a drawback in some protocols, Yao's GC protocol [62] has a constant number of communication rounds and therefore the large number of LowMC rounds does not decrease performance in high-latency networks. Additionally, using the optimizations presented by [22], the large number of linear layer computations can be reduced, bringing the evaluation time of (3) close to (1) and (2). For these reasons, we recommend the use of instance (3) for GC-PSI, which requires 8.2x fewer AND gates than standard AES-128 (6). Thus, we perform all performance evaluations using instance (3). For use cases with small data complexity requirements, we recommend LowMC instance (4), which is a small improvement of 8.3 % in runtime and communication compared to (3). For completeness and direct comparison to AES-128, we also give a variant of LowMC with data complexity of 2^{128} in (5).

[5] https://github.com/LowMC/lowmc/blob/master/determine_rounds.py

	PRF	n	k	m	d	r	#ANDs
(1)	LowMC	128	128	42	2^{64}	13	1,638
(2)	LowMC	128	128	31	2^{64}	13	1,209
(3)	**LowMC**	**128**	**128**	**1**	$\mathbf{2^{64}}$	**208**	**624**
(4)	LowMC	128	128	1	2^{32}	192	576
(5)	LowMC	128	128	1	2^{128}	287	861
(6)	AES-128	128	128	16	2^{128}	10	5,120

Table 2: Comparison of PRF instances for use in the GC-PSI protocol. The recommended instance is highlighted in bold.

4.3 Optimized GC-PSI Protocol

The idea of using Yao's GC protocol for OPRF evaluations was first proposed in [52] and used to construct a PSI protocol in the precomputation setting in [40].

The full protocol description is given in Fig. 1.We propose an optimization that halves the online communication for the OTs (which is the only communication in the online phase). This optimization is of independent interest as it improves the practicality of Yao's GC protocol in arbitrary use cases with precomputation. It is based on the observation that with the Free-XOR technique [42] for Yao's GC protocol [62], the client receives one of the two labels l^0 and $l^1 = l^0 \oplus \Delta$ via OT depending on its input bit, where l^0 is chosen at random and Δ is a random global constant only known by the garbler. A natural consideration would be to replace the real OTs, as used in [40], with correlated OTs (C-OTs) (cf. §3.1). Unfortunately, since the client input is unknown in the base phase, this prevents either the precomputation of the garbled circuits or the OTs. This is because in the online phase when using OT precomputation [8], the random messages r^0 and r^1 obtained by the sender in the base phase need to be swapped

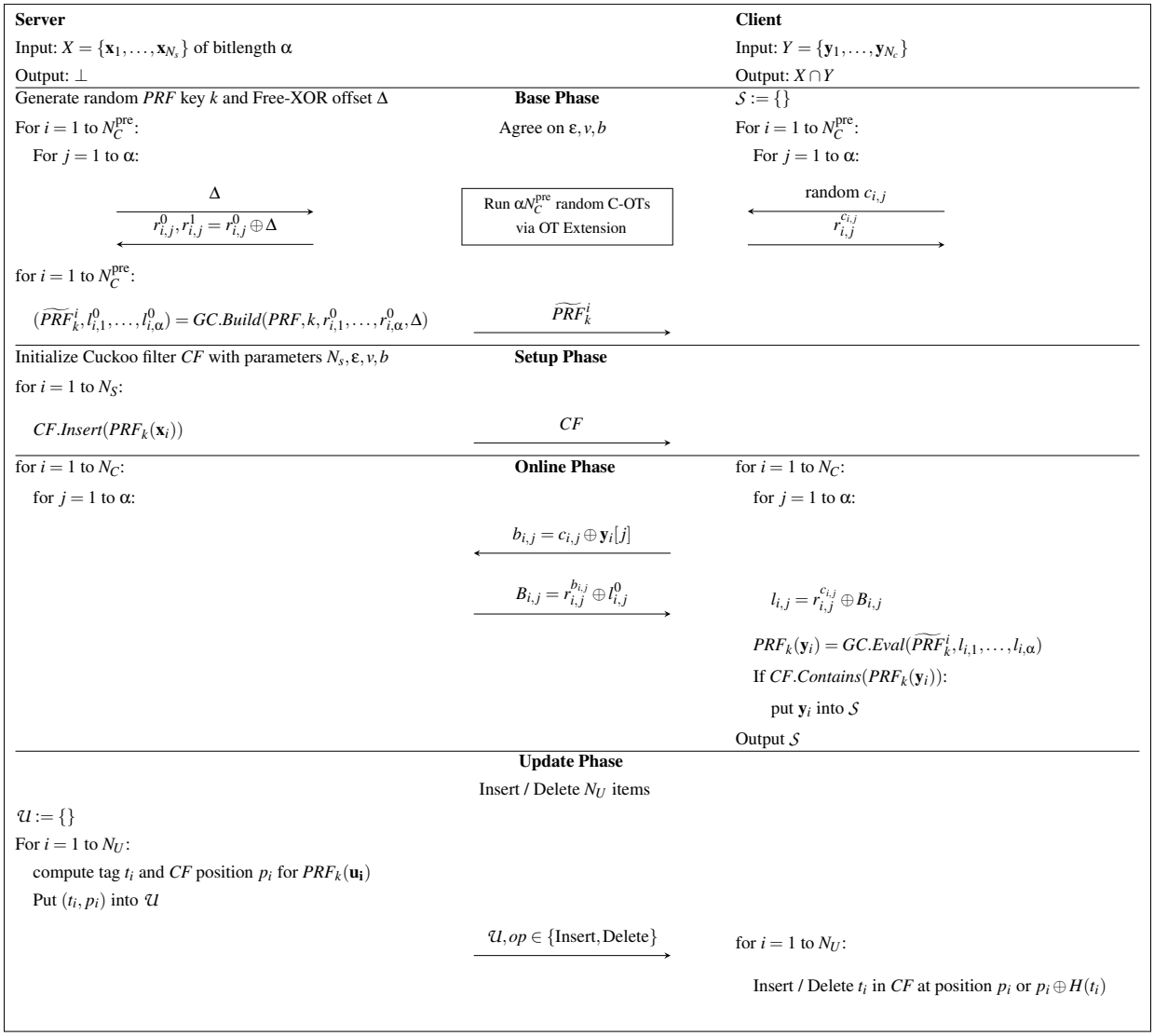


Figure 1: Our optimized GC-PSI protocol (based on [40, 52, 59]). Wire labels are computed as $l_{i,j}^0 = r_{i,j}^0 \oplus \delta_{i,j}$ and $l_{i,j}^1 = l_{i,j}^0 \oplus \Delta$, where the values $\delta_{i,j}$ are chosen at random while building the garbled circuit. $N_C^{\text{pre}} \geq N_C$ denotes the number of precomputed OTs and garbled circuits; the base phase must be repeated before further online phase executions once N_C^{pre} queries are exceeded.

in case the random choice made by the receiver differs from its actual input. Thus, it would be necessary to swap input wire labels in the garbled circuits, which requires recomputing and resending at least the first layer of those circuits.

Our novel precomputation method circumvents this dilemma: In the base phase we run C-OTs via OT extension s.t. the garbler on input Δ learns the random but correlated values r^0 and $r^1 = r^0 \oplus \Delta$, whereas the evaluator upon random choice c learns r^c. For garbling we choose the labels for the input wires of the circuit as $l^0 = r^0 \oplus \delta$ and $l^1 = l^0 \oplus \Delta$. Here, δ is a newly introduced random value that in contrast to Δ is not global but chosen individually for each label pair. In the online phase of the protocol, the evaluator sends a correction bit $b = c \oplus y$ stating whether its random choice c differs from the actual input y. The garbler responds with $B = r^b \oplus l^0$. This way, the evaluator learns either δ or $\delta \oplus \Delta$. It then sets the label for its input to $l = r^c \oplus B$. As one can easily verify for the four possible combinations of random choices c and correction bits b, the evaluator always retrieves the correct label.

The security of the C-OT precomputation is based on the same arguments as standard OT precomputation [8] and since we use a fresh uniformly random δ for each wire label, the resulting wire label is also uniformly random. In other words, we resolve the problem by fixing the wire labels but if necessary swapping the masks required to retrieve the correct label from the initial C-OT result.

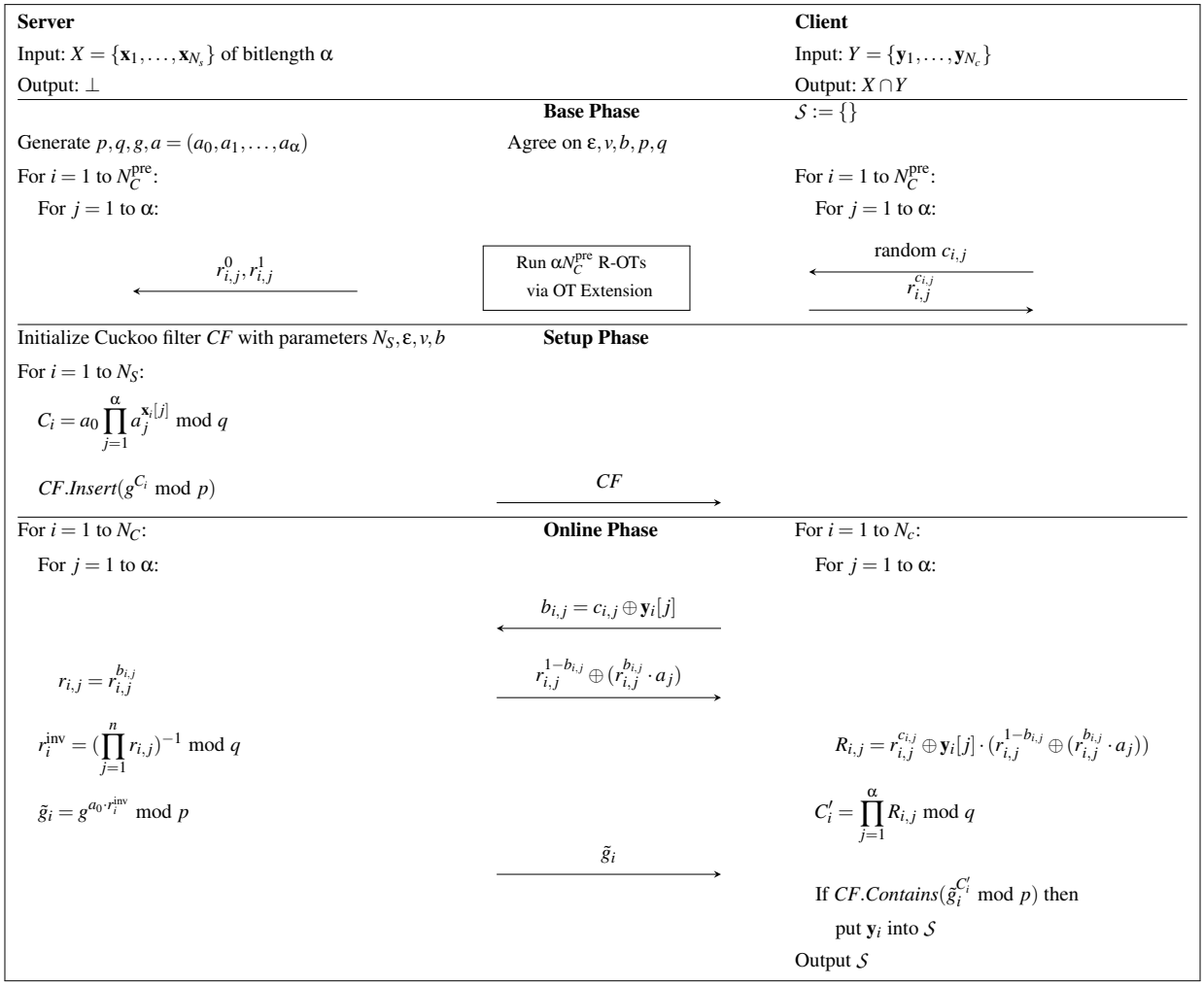


Figure 2: Our optimized NR-PSI protocol (based on [31, 40, 59]). When using a plain finite field, the modulus p is prime, q is a prime divisor of $p-1$, $g \in \mathbb{Z}_p^*$ is of order q, and $a_0, a_1, \ldots, a_\alpha$ as well as $r_{i,j}^0, r_{i,j}^1$ are random numbers in $\mathbb{Z}_q^*$. The update phase is omitted since it is similar to the GC-PSI protocol (cf. Fig. 1), except using the NR-PRF to compute tag t_i and CF position p_i.

4.4 Optimized NR-PSI Protocol

The usage of the Naor-Reingold PRF (NR-PRF) [47] for PSI was first proposed in [31] and the resulting PSI protocol transformed into the precomputation setting in [40]. The NR-PRF for key k and element x_i is defined as

$$f_k(x_i) = g^{a_0 \cdot \prod_{j=1}^{\alpha} a_i^{x_{i,j}}} \mod p, \tag{1}$$

where, when using a plain finite field, p is a prime, q is a prime divisor of $p-1$, $g \in \mathbb{Z}_p^*$ is a generator of order q, $a_0, a_1, \ldots, a_\alpha$ are random numbers in $\mathbb{Z}_q^*$ forming key k, and α is the bitlength of element x_i.

Among all protocols for mobile contact discovery evaluated in [40], NR-PSI is the only protocol besides GC-PSI that can easily be made secure against malicious clients by employing malicious secure OT extensions (cf. §4.5). Furthermore, according to the empirical performance comparison in [40], the NR-PSI protocol causes $\approx$ 30x less communication overhead than GC-PSI without our optimizations. This is why we also consider the NR-PSI protocol in this work and compare it to our optimized GC-PSI implementation in §6.

The full protocol description is given in Fig. 2. We propose an optimization that improves the online communication for OTs by factor 2x. The optimization is based on the observation that in the definitions of [31] the client chooses between a random r and $r \cdot a$ depending on the current bit of its input element. This implies that C-OTs (cf. §3.1) can be used instead of real OTs, thereby sending only one message in the size of the symmetric security parameter instead of the two messages when using the OTe protocols of [3].

Since we use the precomputation form of [40], we propose a novel combination of OT precomputation [8] and C-OT [3]. As in OT precomputation, the client sends a correction bit b stating whether its random choice c in the precomputation

phase equals its real input. Depending on b, the server then decides which of the two random messages obtained during OT precomputation is chosen as r and which is used to mask the correlated message $r \cdot a$ that is sent to the client. Likewise, the client either proceeds with the message obtained during OT precomputation as r or uses this message to unmask the received correlated message.

4.5 Malicious Security

As observed already in [40], the only messages sent by the client in the GC-PSI and NR-PSI protocols are those in the base OT and OT extension protocols as well as the correction bits during the online phase when applying OT precomputation [8]. Therefore, both protocols can easily be made secure against a malicious client by using a maliciously secure OTe protocol such as [4] or [39], together with maliciously secure base OTs such as [50]. As the OT extension contributes only a small percentage to the total runtime of the PSI protocols and today's maliciously secure OTe protocols are only slightly less efficient than the passively secure OT extension of [3], the total runtime of the PSI protocols does not increase by a noticeable amount when replacing the OTe protocols. Please note that enumeration attacks (i.e., querying the server repeatedly with different inputs) are still possible when using our protocols. However, even an ideal functionality for PSI (e.g., a trusted third party) and currently deployed non-private contact discovery methods cannot prevent this. We recommend to employ well-established measures like rate limiting to mitigate such attacks.

The case of a malicious server is different: it could, for example, send wrong wire labels, use wrong circuit descriptions, or send a wrong server set. In general, the client does not reveal the intersection result to the server, so a malicious server can only influence the correctness of the client's computation, but cannot learn any information about the client's items when using maliciously secure OTs. Unfortunately, in most mobile messaging applications, the client sends information about the intersection (most likely even the entire intersection) to the server. This allows a malicious server to learn information about the client's items that are not part of the intersection of the two actual input sets. Therefore, we need to assume a semi-honest server in such scenarios. Preventing malicious behavior on the server side could be done by combining our protocols with a trusted execution environment for hardware-enforced code and remote attestation capabilities s.t. the server's protocol deviation possibilities are restricted to wrong inputs for the Cuckoo filter construction. However, assuming a semi-honest server is reasonable since there are legal requirements and financial incentives for a service provider to behave correctly: once misconduct gets known publicly, users will abandon the malicious service and switch to a more trustworthy alternative.

4.6 Further Extensions

The bottleneck for very large server sets is the communication required to send the Cuckoo filter to the client. For example, a compressed Cuckoo filter for 2^{28} server items with false positive probability $\varepsilon_{max} \approx 2^{-29}$ has a size of ≈ 1 GiB, which is prohibitively large for transmission on mobile network speeds and data plans. For even larger server databases, the protocols eventually become impractical. For example, for a server database with 2^{31} entries, it would be necessary to download a Cuckoo filter of size ≈ 8 GiB. Therefore, we describe how to reduce the overall client-server communication to be logarithmic in the size of the server database. We propose further extensions to increase practicality in App. A.

Combination with Private Information Retrieval (PIR). In their PIR-PSI protocol, Demmler et al. [21] propose the use of multiple non-colluding servers together with a multi-server PIR protocol. Applied to our PSI protocols, the extension works as follows: After the server prepared the Cuckoo filter, it is not transmitted to the client, but to a second non-colluding server instead. Since the Cuckoo filter only contains the results of PRF evaluations, the second server does not learn anything about the items in the main server's set. The client then performs the OPRF evaluation for each of its items with the first server and then runs a multi-server PIR protocol to retrieve the fingerprints stored in the Cuckoo filter.

The communication complexity for the multi-server PIR lookup is $O(\kappa \log n)$, where κ is the symmetric security parameter and n the size of the server database [14, 21]. Since the overall client-server communication therefore is logarithmic and not linear in the size of the server database, our protocols are expected to remain practical even for server databases with more than a billion items. In practice, the remaining challenge for messaging services is to find a trustworthy partner operating the second PIR server while at the same time making it credible to users that no collusion is happening.

5 Android Implementation

To demonstrate the feasibility of our optimized PSI protocols for performing private contact discovery on mobile devices, we provide implementations for smartphones running on Android.[6] Previous works [34, 40] presented experiments on dedicated mobile devices, but the performance of these implementations was not sufficient for real-world usage. For example, the Java implementation of [40], which is based on the ObliVM framework [43], takes more than a second to evaluate a single garbled AES-128 circuit. In our implementation, we make use of native C/C++ code support in Android and also use hardware acceleration for cryptographic operations available in modern smartphones. More precisely, native

[6] https://contact-discovery.github.io

AES-128 instructions are used both as a PRNG and during the creation and evaluation of the garbled circuit. These features allow our implementation to reach truly practical performance. Compared to the Java-based implementation of [40], we evaluate a garbled AES-128 circuit more than 1,000x faster.

5.1 Base OTs and OT Extension

For performing base OTs, we use the OT protocol of Chou and Orlandi [20] with the additional verification step proposed by Doerner et al. [26]. Together with the (C-)OT extension protocol of Keller, Orsini, and Scholl [39], this results in a maliciously secure protocol (cf. [26]).

Our OT implementation is based on `libOTe` by Rindal [60], which is heavily optimized for the x86 architecture. Thus, we ported large parts of the library to the ARMv8 architecture to achieve high performance on mobile devices. At the same time, we kept the library compatible with its x86 counterpart to facilitate natural development of client-server applications.

5.2 GC-PSI Implementation

For the GC-PSI protocol, we implement Yao's GC protocol (cf. §3.2) with Free-XOR [42] and Half-Gates [63], resulting in no communication for XOR-gates and two wire labels of κ bits each per AND gate, where $\kappa = 128$ is the symmetric security parameter.

For creating and evaluating the garbled tables, the most efficient choice today is fixed-key AES [11], mainly due to the hardware support for AES that is widespread in modern x86 CPUs. The ARM Cryptography Extensions (CE) introduced in the ARMv8 architecture similarly provide hardware instructions for AES, SHA-1, and SHA-2 variants, resulting in AES speedups of factor 35x compared to a standard AES software implementation. This allows us to also use fixed-key AES [11] for garbling in our implementation.[7] Additionally, the ARMv8 architecture provides instructions for vector operations on 128-bit registers (the so-called NEON instruction set), which we use to efficiently work with 128-bit wire labels. In Tab. 7 in App. B, we demonstrate the wide availability of ARM CE in most recent smartphone processors.

5.3 NR-PSI Implementation

For implementing the NR-PSI protocol, we use the modified `libOTe` version described in §5.1 for C-OT precomputation as well as the `GNU GMP`[8] library for modular arithmetic operations and the `MIRACL`[9] library for instantiating the protocol with elliptic curve P-256. The advantage of instantiating the NR-PSI protocol with ECC instead of using a plain finite field with comparable security parameters is that the size of the values $\tilde{g}_i$ transferred during the online phase (cf. Fig. 2) is reduced by factor 8x. Also, computationally expensive modular exponentiations are replaced with point multiplications. We refer to this variant as ECC-NR-PSI in the following. All libraries are compiled specifically for the ARMv8 architecture.

[7] As recently reported by [29], many secure computation implementations use fixed-key AES incorrectly. However, according to [29], our instantiation for garbling following the definitions of [63] is not affected. In contrast, `libOTe` [60] is currently vulnerable. The suggested fixes however are not expected to result in a significant negative performance impact [29].

[8] https://gmplib.org

[9] https://github.com/miracl/MIRACL

6 Performance Evaluation

We empirically evaluate the performance of our optimized GC-PSI and NR-PSI protocols and compare them to other unbalanced PSI protocols from the literature.

Benchmark Settings. For easy comparison to related work, we choose similar sizes for the server's and the client's set: $N_s \in \{2^{20}, 2^{24}, 2^{26}, 2^{28}\}$ and $N_c \in \{1, 2^8, 2^{10}\}$. Here, $N_c = 1$ represents the case where a client wants to check a new contact. All items have a bitlength of $\alpha = 128$. We instantiate all primitives and protocols with 128-bit security.

In all of our experiments, the sever is equipped with an Intel Core^{TM} i7-4600U CPU @ 2.6 GHz and 16 GiB of RAM. The client is a Google Pixel XL 2 smartphone with a Snapdragon 835 CPU @ 2.45 GHz and 4 GiB of RAM. We consider two network settings: (i) an IEEE 802.11ac WiFi connection with $\approx$ 230 Mbit/s down-/upload and 70 ms RTT and (ii) a mobile LTE connection with 42 Mbit/s download ($S \rightarrow C$), 4 Mbit/s upload ($S \leftarrow C$), and 80 ms RTT.

Note that the LTE network speeds are real-world parameters and exhibit a significant difference in the down- and upload rates. This is common in commercially available data plans and often not taken into account in previous evaluations.

6.1 GC-PSI and NR-PSI Protocol

The runtime and communication costs for the base, setup, and online phase of our protocols are shown in Tab. 3, Tab. 4, and Tab. 5, respectively, and are averaged over 100 executions (except for the setup phase, where we chose 10 or less executions due to the larger runtime). We use LowMC instance (3) from Tab. 2 for the evaluation. In all tests, only a single thread was used for both the server and the client. Since all phases of our protocols can be parallelized trivially, we expect a near-linear speedup when using multiple threads, except in situations where the bottleneck is network bandwidth. Furthermore, note that in the base and online phases of the GC-PSI protocol, only one party actually performs the computationally expensive task of garbling or evaluating the circuit. Therefore, if both parties are ready, the base and online phases of the GC-PSI protocol can be interleaved in a pipelined fashion, where the server sends the garbled circuits and the client evaluates them as soon as parts of them are available. This

Parameters		Time [s]		Comm. [MiB]	
N_c^{pre}	Protocol	WiFi	LTE	$S \rightarrow C$	$S \leftarrow C$
2^{10}	AES-GC-PSI	7.14	38.98	162.52	2.02
	LowMC-GC-PSI	1.85	6.57	22.01	2.02
	ECC-NR-PSI	**0.61**	**4.21**	**0.01**	**1.99**

Table 3: Base phase of our PSI protocols. Precomputation for checking N_c^{pre} client contacts. Best results marked in bold.

Parameters		Server Setup [s]	Transmission [s]		Comm. [MiB]
N_s	Protocol		WiFi	LTE	$S \rightarrow C$
2^{28}	AES-GC-PSI	**23.94**			
	LowMC-GC-PSI	1,869.13	32.66	211.30	1072
	ECC-NR-PSI	52,332.38			
2^{26}	AES-GC-PSI	**4.87**			
	LowMC-GC-PSI	467.29	8.13	52.55	268
	ECC-NR-PSI	12,787.79			
2^{24}	AES-GC-PSI	**1.12**			
	LowMC-GC-PSI	116.66	2.13	13.05	67
	ECC-NR-PSI	3,297.96			
2^{20}	AES-GC-PSI	**0.06**			
	LowMC-GC-PSI	7.27	0.25	0.63	4.19
	ECC-NR-PSI	241.54			

Table 4: Setup phase of our PSI protocols. Server setup run once for *all* clients. The Cuckoo filter parameters are set as described in §4.1 ($\varepsilon_{max} = 2^{-29.4}, v = 32, b = 3$). Best results marked in bold. Note that the size of the client set does not influence the runtime of the setup phase and the client does not send any data during the setup phase in any protocol.

method can reduce the runtime of the combined base and online phase to the runtime of the slower phase.

We observe that using LowMC instead of AES in the GC-PSI protocol leads to 7.4x less communication and thus to a much smaller runtime in the base phase, while the online phase of both protocol versions is very comparable. Only during the one-time setup phase, the AES version is more efficient due to AES-NI instructions. Using a hardware-accelerated implementation of LowMC could reduce this runtime close to the one of AES, but we again stress that the setup phase is a one-time cost. This confirms our choice of LowMC over AES as the PRF in GC-PSI.

ECC-NR-PSI is the most efficient protocol during the base phase since it does not send garbled circuits to the client: compared to the LowMC version of GC-PSI, it requires 12x less communication. The ECC-NR-PSI online phase is slightly slower than both GC-PSI protocols, while being the fastest for a single item. The one-time setup phase of the ECC-NR-PSI protocol is much slower than both GC-PSI protocol versions due to elliptic curve operations.

6.2 Comparison with Related Work

We now highlight differences to other works in the literature and compare our optimized GC- and NR-PSI protocols and implementations to other unbalanced PSI implementations available for Android in Tab. 6. Comparisons with implementations for the x86 architecture are given in App. D.

Parameters		Time [s]		Comm. [KiB]	
N_c	Protocol	WiFi	LTE	$S \rightarrow C$	$S \leftarrow C$
2^{10}	AES-GC-PSI	**1.43**	**1.86**	**2,048**	**16.00**
	LowMC-GC-PSI	1.71	2.02	**2,048**	**16.00**
	ECC-NR-PSI	2.31	2.32	4,147	**16.00**
2^8	AES-GC-PSI	**0.34**	**0.47**	**512**	**4.00**
	LowMC-GC-PSI	0.37	0.48	**512**	**4.00**
	ECC-NR-PSI	0.61	0.61	1,037	**4.00**
1	AES-GC-PSI	0.03	0.03	**2.00**	**0.02**
	LowMC-GC-PSI	0.04	0.05	**2.00**	**0.02**
	ECC-NR-PSI	**0.01**	**0.02**	4.06	0.04

Table 5: Online phase of our PSI protocols. Best results marked in bold. The influence of the server set size on runtime and communication is negligible and therefore not listed.

Chen et al. [18, 19]. The protocols of [18, 19] for unbalanced PSI are based on leveled fully homomorphic encryption (FHE). They both work as follows: the client encrypts all its items and sends them to the server, which then computes the intersection under encryption with all of its own items and returns the result in encrypted form. The client can then decrypt the received ciphertexts to find the intersection.

The protocol in [19] is only defined for 32 bit strings, a limitation that stems from the parameter choice of the FHE scheme. Since the universe of possible items is larger than 2^{32} in the use case of contact discovery, we exclude this protocol from further comparisons. However, this limitation was lifted in the subsequent work [18] where arbitrary length items are supported. The benefits of [18] compared to our protocols are that the client is not required to store any data and that the total communication is sublinear in the size of the server database. For example, for $N_s = 2^{28}$, the total communication in the protocol of [18] is only 18.4 MB.

However, there is a huge computational overhead during the online phase of the protocol: even on a high-end server it takes more than 12 s on 32 threads to compute the intersection with $N_c = 1024$ client elements. Unfortunately, the online phase needs to be repeated whenever there are updates on client or server side. Also, due to the employed FHE batching optimizations, the runtime for a single item is almost equal to the runtime for thousands of items. Assuming that each of the $N_s = 2^{28}$ registered clients runs one update per day, this would require the service provider to pay for $2^{28} \cdot 12.1 \cdot 32 \approx 28.9$ million core hours every day. In contrast, the online phases of our protocols run in ≈ 2 s for $N_c = 1024$ in the WiFi setting on a single-threaded smartphone and require no cryptographic operations on server side. The evaluation of [18] was performed on two servers with Intel Xeon CPUs in a 10 Gbit/s local network. Therefore, it is also unclear how the FHE encryption and decryption routines perform in a mobile setting on real smartphones.

Resende and de Freitas Aranha [59]. In [59], the authors present implementation improvements for the PSI protocol of [7]. For each element in the client's set, they perform 3

Parameters N_s	Parameters N_c	PSI Protocol	Base + Online Time [s] WiFi	Base + Online Time [s] LTE	Communication [MiB] $S \rightarrow C$	Communication [MiB] $S \leftarrow C$	Setup Communication / Client Storage [MiB]	Setup Transfer [s] WiFi	Setup Transfer [s] LTE	Server Setup [s]
2^{28}	1,024	AES-GC-PSI [40]	1,507.73	2,742.66	177.23	4.00	1,380.25	42.05	272.06	**26.70**
		NR-PSI [40]	171.23	221.20	64.25	2.02	1,380.25	42.05	272.06	194,130.21
		LowMC-GC-PSI (Ours)	3.54	8.59	22.01	2.02	**1,072.00**	**32.66**	**211.30**	1,869.13
		ECC-NR-PSI (Ours)	**2.92**	**6.53**	**4.07**	**2.00**	**1,072.00**	**32.66**	**211.30**	52,332.38
	1	AES-GC-PSI [40]	1.53	2.95	0.18	0.02	1,380.25	42.05	272.06	**26.70**
		NR-PSI [40]	0.17	0.21	0.06	**0.01**	1,380.25	42.05	272.06	194,130.21
		LowMC-GC-PSI (Ours)	0.17	0.18	0.04	0.02	**1,072.00**	**32.66**	**211.30**	1,869.13
		ECC-NR-PSI (Ours)	**0.13**	**0.13**	**0.01**	**0.01**	**1,072.00**	**32.66**	**211.30**	52,332.38
2^{24}	1,024	AES-GC-PSI [40]	1,507.73	2,742.66	177.23	4.00	86.26	2.74	16.80	**1.18**
		NR-PSI [40]	171.23	221.20	64.25	2.02	86.26	2.74	16.80	12,174.40
		LowMC-GC-PSI (Ours)	3.54	8.59	22.01	2.02	**67.00**	**2.13**	**13.05**	116.66
		ECC-NR-PSI (Ours)	**2.92**	**6.53**	**4.07**	**2.00**	**67.00**	**2.13**	**13.05**	3,297.96
	1	AES-GC-PSI [40]	1.53	2.95	0.18	0.02	86.26	2.74	16.80	**1.18**
		NR-PSI [40]	0.17	0.21	0.06	**0.01**	86.26	2.74	16.80	12,174.40
		LowMC-GC-PSI (Ours)	0.17	0.18	0.04	0.02	**67.00**	**2.13**	**13.05**	116.66
		ECC-NR-PSI (Ours)	**0.13**	**0.13**	**0.01**	**0.01**	**67.00**	**2.13**	**13.05**	3,297.96
2^{20}	1,024	AES-GC-PSI [40]	1,507.73	2,742.66	177.23	4.00	5.39	0.32	0.81	**0.05**
		NR-PSI [40]	171.23	221.20	64.25	2.02	5.39	0.32	0.81	758.40
		LowMC-GC-PSI (Ours)	3.54	8.59	22.01	2.02	**4.19**	**0.25**	**0.63**	7.27
		ECC-NR-PSI (Ours)	**2.92**	**6.53**	**4.07**	**2.00**	**4.19**	**0.25**	**0.63**	241.54
	1	AES-GC-PSI [40]	1.53	2.95	0.18	0.02	5.39	0.32	0.81	**0.05**
		NR-PSI [40]	0.17	0.21	0.06	**0.01**	5.39	0.32	0.81	758.40
		LowMC-GC-PSI (Ours)	0.17	0.18	0.04	0.02	**4.19**	**0.25**	**0.63**	7.27
		ECC-NR-PSI (Ours)	**0.13**	**0.13**	**0.01**	**0.01**	**4.19**	**0.25**	**0.63**	241.54

Table 6: Comparison of PSI protocols with smartphone implementations. Numbers for protocols of [40] are obtained by running their implementations in our benchmarking environment. In all tests $N_c^{\text{pre}} = N_c$. Best in class marked in bold.

point multiplications and transmit 2 group elements. This results in a lower communication than our approaches (64 B for 2 group elements vs. 22 KiB per garbled circuit vs. 6 KiB per item in NR-PSI). However, one major contribution of [59] is a significant optimization of the GLS-254 curve for x86 CPUs. It is therefore unclear how their protocol performs on smartphones with ARMv8-A hardware. Furthermore, their Cuckoo filters parameters allow for a false positive probability that is too high for real-world deployment (cf. §4.1). Finally, their protocol assumes semi-honest adversaries, and while a maliciously secure variant [38] of their basic protocol exists, its performance has not yet been evaluated.

Kiss et al. [40]. In [40], the authors consider various semi-honest PSI protocols, from which their GC-PSI and NR-PSI protocols are the foundation of our work. Their Android implementation (in pure Java) takes about 1.5 s for a single oblivious AES evaluation in their GC-PSI protocol. The authors therefore conclude that instead their ECC-DH-PSI protocol is most suited for the mobile use case since the evaluation time for a single item is 23 ms. However, both of our optimized protocols with security against malicious clients are more than competitive with an evaluation time of less than 2 ms for a single item. For $N_c = 1024$ client elements, the combined base and online time of our optimized GC- and NR-PSI protocols improves by more than a factor of 300x and 30x, respectively, compared to the unoptimized semi-honest implementations of [40] in both the WiFi and the LTE network setting. Also, the total communication during the base and online phase improves by factors 7.5x and 10.9x compared to the respective GC- and NR-PSI protocols of [40].

7 Conclusion

Our native implementations of our optimized NR- and GC-PSI protocols are two almost equivalently outstanding solutions for large-scale mobile private contact discovery with security against malicious clients. The Signal developers stated that to actually deploy PSI-based contact discovery, it would need to be able to handle a server database with 1 billion users while address books are assumed to contain up to 10,000 contacts. In terms of latency, lookups are required to take less than 2 s, while in terms of throughput a single core should be able to handle 1,600 contacts per second. Clearly, we cannot meet these demanding requirements yet. Therefore, as part of future work, we suggest to implement and evaluate our proposed extensions (especially the combination with PIR) to take the next important steps towards real-world deployment.

Acknowledgments

This work was co-funded by the DFG as part of project E4 within the CRC 1119 CROSSING and project A.1 within the RTG 2050 “Privacy and Trust for Mobile Users”, by the BMBF and the HMWK within CRISP, and by the European Union’s Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR). Daniel Kales has been supported by iov42 Ltd.

References

[1] WhatsApp Legal Info. https://www.whatsapp.com/legal, 2019.

[2] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *CCS*, pages 535–548. ACM, 2013.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. In *EUROCRYPT*, volume 9056 of *LNCS*, pages 673–701. Springer, 2015.

[5] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer Extensions. *Journal of Cryptology*, 30(3):805–858, 2017.

[6] N. Asokan, Alexandra Dmitrienko, Marcin Nagy, Elena Reshetova, Ahmad-Reza Sadeghi, Thomas Schneider, and Stanislaus Stelle. CrowdShare: Secure Mobile Resource Sharing. In *ACNS*, volume 7954 of *LNCS*, pages 432–440. Springer, 2013.

[7] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes. In *CCS*, pages 691–702. ACM, 2011.

[8] Donald Beaver. Precomputing Oblivious Transfer. In *CRYPTO*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.

[9] Donald Beaver. Correlated Pseudorandomness and the Complexity of Private Computations. In *STOC*, pages 479–488. ACM, 1996.

[10] Donald Beaver, Silvio Micali, and Phillip Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *STOC*, pages 503–513. ACM, 1990.

[11] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient Garbling from a Fixed-Key Blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society, 2013.

[12] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[13] Joan Boyar and René Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In *Symposium on Experimental Algorithms*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *CCS*, pages 1292–1303. ACM, 2016.

[15] Alexander Breslow and Nuwan Jayasena. Morton Filters: Faster, Space-Efficient Cuckoo Filters via Biasing, Compression, and Decoupled Logical Sparsity. *Proceedings of the VLDB Endowment (PVLDB)*, 11(9):1041–1055, 2018.

[16] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security*, pages 991–1008. USENIX Association, 2018.

[17] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure Outsourced Garbled Circuit Evaluation for Mobile Devices. In *USENIX Security*, pages 289–304. USENIX Association, 2013.

[18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *CCS*, pages 1223–1237. ACM, 2018.

[19] Hao Chen, Kim Laine, and Peter Rindal. Fast Private Set Intersection from Homomorphic Encryption. In *CCS*, pages 1243–1255. ACM, 2017.

[20] Tung Chou and Claudio Orlandi. The Simplest Protocol for Oblivious Transfer. In *LATINCRYPT*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.

[21] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling Private Contact Discovery. *PoPETs*, 2018(4):159–178, 2018.

[22] Itai Dinur, Daniel Kales, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. Linear Equivalence of Block Ciphers with Partial Non-Linear Layers: Application to LowMC. In *EUROCRYPT*, volume 11476 of *LNCS*, pages 343–372. Springer, 2019.

[23] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized Interpolation Attacks on LowMC. In *ASIACRYPT*, volume 9453 of *LNCS*, pages 535–560. Springer, 2015.

[24] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-Order Cryptanalysis of LowMC. In *ICISC*, volume 9558 of *LNCS*, pages 87–101. Springer, 2015.

[25] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *PKC*, volume 3386 of *LNCS*, pages 416–431. Springer, 2005.

[26] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure Two-party Threshold ECDSA from ECDSA Assumptions. In *IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society, 2018.

[27] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In *Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 75–88. ACM, 2014.

[28] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*, volume 3378 of *LNCS*, pages 303–324. Springer, 2005.

[29] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers. *IACR Cryptology ePrint Archive*, 2019:074, 2019. https://ia.cr/2019/074.

[30] Carmit Hazay and Yehuda Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *TCC*, volume 4948 of *LNCS*, pages 155–175. Springer, 2008.

[31] Carmit Hazay and Yehuda Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *Journal of Cryptology*, 23(3):422–456, 2010.

[32] Wilko Henecka and Thomas Schneider. Faster secure two-party computation with less memory. In *ASIACCS*, pages 437–446. ACM, 2013.

[33] Kashmir Hill. Facebook recommended that this psychiatrist's patients friend each other. https://splinternews.com/facebook-recommended-that-this-psychiatrists-patients-f-1793861472, 2016.

[34] Yan Huang, Peter Chapman, and David Evans. Privacy-Preserving Applications on Smartphones. In *HotSec*, pages 4–4. USENIX Association, 2011.

[35] Russell Impagliazzo and Steven Rudich. Limits on the Provable Consequences of One-way Permutations. In *STOC*, pages 44–61. ACM, 1989.

[36] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[37] Stanislaw Jarecki and Xiaomin Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, volume 5444 of *LNCS*, pages 577–594. Springer, 2009.

[38] Stanislaw Jarecki and Xiaomin Liu. Fast Secure Computation of Set Intersection. In *SCN*, volume 6280 of *LNCS*, pages 418–435. Springer, 2010.

[39] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively Secure OT Extension with Optimal Overhead. In *CRYPTO*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

[40] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private Set Intersection for Unequal Set Sizes with Mobile Applications. *PoPETs*, 2017(4):177–197, 2017.

[41] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *CCS*, pages 818–829. ACM, 2016.

[42] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[43] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A Programming Framework for Secure Computation. In *IEEE Symposium on Security and Privacy*, pages 359–376. IEEE Computer Society, 2015.

[44] Moxie Marlinspike. The Difficulty Of Private Contact Discovery. https://signal.org/blog/contact-discovery, 2014.

[45] Moxie Marlinspike. Technology Preview: Private Contact Discovery for Signal. https://signal.org/blog/private-contact-discovery, 2017.

[46] Benjamin Mood, Debayan Gupta, Kevin R. B. Butler, and Joan Feigenbaum. Reuse It Or Lose It: More Efficient Secure Computation Through Reuse of Encrypted Values. In *CCS*, pages 582–596. ACM, 2014.

[47] Moni Naor and Omer Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. *Journal of the ACM*, 51(2):231–262, 2004.

[48] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. In *Annual European Symposium on Algorithms*, volume 2161 of *LNCS*, pages 121–133. Springer, 2001.

[49] Panagiotis Papadopoulos, Antonios A. Chariton, Elias Athanasopoulos, and Evangelos P. Markatos. Where's Wally?: How to Privately Discover your Friends on the Internet. In *ASIACCS*, pages 425–430. ACM, 2018.

[50] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.

[51] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*, pages 515–530. USENIX Association, 2015.

[52] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure Two-Party Computation Is Practical. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[53] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient Circuit-based PSI with Linear Communication. In *EUROCRYPT*, volume 11476 of *LNCS*, pages 122–153. Springer, 2019.

[54] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*, volume 10822 of *LNCS*, pages 125–157. Springer, 2018.

[55] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster Private Set Intersection Based on OT Extension. In *USENIX Security*, pages 797–812. USENIX Association, 2014.

[56] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable Private Set Intersection Based on OT Extension. *ACM Transactions on Privacy and Security*, 21(2):7:1–7:35, 2018.

[57] Michael Rabin. How to Exchange Secrets with Oblivious Transfer. In *Technical Report TR-81*. Aiken Computation Laboratory: Harvard University, 1981.

[58] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of Low-Data Instances of Full LowMCv2. *IACR Transactions on Symmetric Cryptology*, 2018(3):163–181, 2018.

[59] Amanda Cristina Davi Resende and Diego de Freitas Aranha. Faster Unbalanced Private Set Intersection. In *FC*, LNCS. Springer, 2018.

[60] Peter Rindal. libOTe: A fast, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[61] Statista. Most Popular Global Mobile Messenger Apps. https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps, 2019.

[62] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*, pages 162–167. IEEE, 1986.

[63] Samee Zahur, Mike Rosulek, and David Evans. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

A Protocol Extensions

We propose further extensions for improving practicality.

Combination with FHE Protocols. Protocols for unbalanced PSI based on fully homomorphic encryption (FHE), e.g., [18], are computationally expensive and thus much slower during the online phase than our protocols (cf. §6.2). However, their advantage is that the total amount of communication is sublinear in the size of the server database. When clients install a new messaging application and are not connected to a high-speed WiFi network, such FHE-based protocols likely produce faster contact discovery results, which leads to higher user satisfaction. Thus, we recommend the following hybrid use of contact discovery protocols: Directly after installation of a mobile messaging application, a FHE-based protocol (e.g., [18]) is used to perform the initial contact discovery. Then, while the phone is charging overnight and is connected to a WiFi network, the base and setup phase of one of our protocols is performed. This leads to very efficient online phases for future protocol runs, which are performed regularly when updates on client or server side happen (potentially over mobile data plans where communication matters). See also §6.2 for a more detailed comparison between FHE-based unbalanced PSI protocols and our work.

Dedicated Server for Cuckoo Filter Membership Tests. In many scenarios, a large number of clients is part of a single organization. For example, consider the mobile malware detection scenario discussed in [40], where all applications installed on a client's smartphone are checked against a database of malicious applications. When employing such a malware detection service in an enterprise context, a company usually buys a volume license for all of its employees.

To reduce the overall data communication, the company could host a dedicated server which would receive the large encrypted database of server items represented as a compressed Cuckoo filter once. If a client then wants to compute the intersection between installed and malicious applications, it only communicates with the malware detection service provider to

perform OPRF evaluations and then hands off the encrypted items to the trusted company server, which performs the set intersection on behalf of the clients and reports back the result. Since this trusted server does not have knowledge of the PRF key, it cannot directly deduce which items the client holds.

However, since the OPRF result is deterministic when using the same secret key, the trusted server can learn when multiple clients request the same item. Furthermore, it could interact with the malware detection service provider itself to obtain encryptions of known items, which it can compare to the encrypted items of the clients. However, this kind of leakage can be argued to be acceptable in many settings, such as the company-internal setting mentioned above.

Partitioning the Database. A simple solution to reduce the required communication during the setup phase is to partition the server database s.t. clients only download Cuckoo filters relevant for the contacts in their address book (for example w.r.t. number prefixes, states, countries, or regions).

Assuming that the majority of users has contacts in only very few such partitions, this approach leads to practical data transmission sizes even for services with billions of users. In the worst case (i.e., a user has contacts in all partitions or prefers to leak no information at all), multiple runs of our protocols can cover the worldwide user base.

However, this solution presents a significant performance / privacy trade-off since clients leak information about their social graph. For example, intelligence agencies might find it suspicious if US citizens evidently have contacts in middle eastern countries. How severe the privacy of users is threatened also depends on how fine-grained the chosen partitions are: if they are too small, it might even be possible to identify an individual just by observing Cuckoo filter downloads.

B ARM Cryptography Extensions (CE)

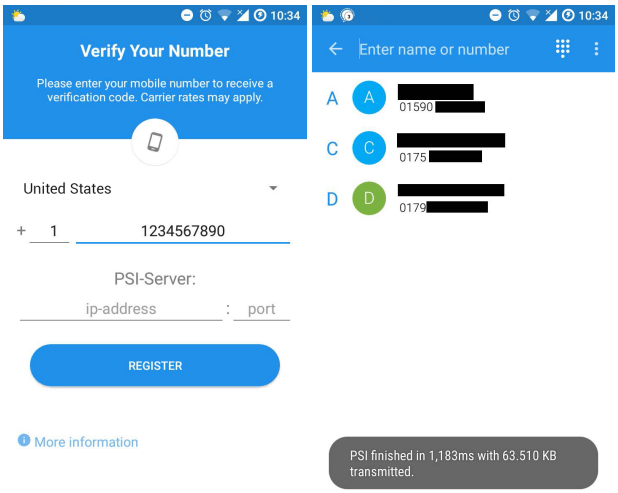
The wide availability of the ARM Cryptography Extensions (CE) in modern smartphone processors is highlighted in Tab. 7.

System-on-a-Chip (SoC)	Example Smartphones and Tablets	CE
Apple A4, A5, A6	iPhone 4, iPad, iPad 2, iPhone 5	✗
Apple A7, A8, A9	iPhone (5s,6), iPad Air, iPad mini 2	✓
Apple A10, A11, A12	iPhone (7,8,X,Xs), iPad (2018), iPad Pro	✓
Snapdragon 801	HTC One (E8), OnePlus One	✗
Snapdragon 805	Galaxy S5+, Nexus 6	✗
Snapdragon 808	Nexus 5X, LG G4, Moto X Style	✓
Snapdragon 810	OnePlus 2, Nexus 6P, Sony Xperia Z5	✓
Snapdragon 820	OnePlus 3, Galaxy S7, LG G5	✓
Snapdragon 821	Google Pixel (XL), LG G6	✓
Snapdragon 835	Google Pixel 2 (XL), Galaxy S8	✓
Snapdragon 845	OnePlus 6, Galaxy S9, Sony Xperia Z2	✓

Table 7: Availability of ARM Cryptography Extensions (CE) in modern smartphone and tablet systems-on-a-chip (SoCs).

C Signal Integration Demonstrator

As a proof-of-concept, we modified the client application of the open-source messenger Signal to perform contact discovery using our PSI protocols. To be able to run the modified client with the official servers, the integration works as follows: Whenever Signal triggers the contact discovery routine, we run one of the PSI protocols with our own PSI server[10]. The resulting matches are then used as input for the unmodified Signal contact discovery routine. This way, the official Signal server only learns the hashes of phone numbers which are already registered to the service. Our changes to the user interface of the Android version of the Signal application are depicted in Fig. 3.



(a) Signal registration. (b) Contact discovery result.

Figure 3: Screenshots of our prototype integration into the open-source messenger Signal.

D Comparison of Unbalanced PSI Protocols on the x86 Architecture

The goal of our paper is to provide efficient private contact discovery for mobile messaging applications via improved unbalanced PSI protocols with implementations optimized for smartphones. Therefore, we focus our implementation and evaluation efforts on the mobile use case and perform our experiments on real smartphones with ARMv8 architecture. However, to present the complete picture, we give a comparison to protocols for unbalanced PSI running on x86 hardware and communicating in a local network in Tab. 8.

[10] In practice, this PSI server would be run by Signal and use the actual database of Signal users.

Parameters N_s	Parameters N_c	Protocol	Online Time [s]	Online Communication [MiB]	Setup Communication / Client Storage [MiB]	Server Setup [s]
2^{28}	1,024	[59]	∗0.16	0.07	806	∗182
		[18]	∗12.10	18.57	0	∗4,628
		LowMC-GC-PSI (Ours)	0.93	24.01	1,072	1,869
		ECC-NR-PSI (Ours)	1.34	6.06	1,072	52,332
2^{24}	11,041	[59]	0.71	0.67	48	342
		[19]	44.70	23.20	0	71
		[18]	20.10	41.48	0	656
		LowMC-GC-PSI (Ours)	12.51	258.79	67	117
		ECC-NR-PSI (Ours)	11.94	65.24	67	3,298
	5,535	[59]	0.35	0.34	48	342
		[19]	40.10	20.10	0	64
		[18]	22.01	16.39	0	806
		LowMC-GC-PSI (Ours)	5.63	129.73	67	117
		ECC-NR-PSI (Ours)	5.93	32.71	67	3,298
2^{20}	11,041	[59]	0.71	0.67	3	22
		[19]	6.40	11.50	0	6.4
		[18]	4.49	14.34	0	43
		LowMC-GC-PSI (Ours)	12.51	258.79	4.2	7.3
		ECC-NR-PSI (Ours)	11.94	65.24	4.2	242
	5,535	[59]	0.35	0.34	3	22
		[19]	4.30	5.60	0	4.3
		[18]	4.23	11.50	0	43
		LowMC-GC-PSI (Ours)	5.63	129.73	4.2	7.3
		ECC-NR-PSI (Ours)	5.93	32.71	4.2	242

Table 8: Comparison of unbalanced PSI protocols in the LAN setting (10 Gbit/s, 0.02 ms RTT) on PC hardware (x86 architecture). Numbers for other protocols are taken from [18]. All numbers are from single-core executions, except those marked with ∗, which was an execution with 32 cores on the server side and 4 cores on the client side. The bit length α of all items is 128, except for [19], where $\alpha = 32$ due to limitations of the protocol.

C DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop (WiSec'21)

[HHS+21] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop"**. In: *14. ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/893. ACM, 2021, pp. 371–373. Appendix C.

https://doi.org/10.1145/3448300.3468252

DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop

Alexander Heinrich
TU Darmstadt, Germany
aheinrich@seemoo.de

Matthias Hollick
TU Darmstadt, Germany
mhollick@seemoo.de

Thomas Schneider
TU Darmstadt, Germany
schneider@encrypto.cs.tu-darmstadt.de

Milan Stute
TU Darmstadt, Germany
mstute@seemoo.de

Christian Weinert
TU Darmstadt, Germany
weinert@encrypto.cs.tu-darmstadt.de

ABSTRACT

Apple's file-sharing service AirDrop leaks phone numbers and email addresses by exchanging vulnerable hash values of the user's own contact identifiers during the authentication handshake with nearby devices. In a paper presented at USENIX Security'21, we theoretically describe two attacks to exploit these vulnerabilities and propose "PrivateDrop" as a privacy-preserving drop-in replacement for Apple's AirDrop protocol based on private set intersection.

In this demo, we show how these vulnerabilities are efficiently exploitable via Wi-Fi and physical proximity to a target. Privacy and security implications include the possibility of conducting advanced spear phishing attacks or deploying multiple "collector" devices in order to build databases that map contact identifiers to specific locations. For our proof-of-concept, we leverage a custom rainbow table construction to reverse SHA-256 hashes of phone numbers in a matter of milliseconds. We discuss the trade-off between success rate and storage requirements of the rainbow table and, after following responsible disclosure with Apple, we publish our proof-of-concept implementation as "AirCollect" on GitHub.

CCS CONCEPTS

• **Networks** → **Network protocols**; • **Security and privacy** → **Mobile and wireless security**.

KEYWORDS

privacy, personal information, hashing, rainbow table, iOS, macOS

ACM Reference Format:
Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. 2021. DEMO: AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop. In *14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), June 28–July 2, 2021, Abu Dhabi, United Arab Emirates.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3448300.3468252

WiSec '21, June 28–July 2, 2021, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-8349-3/21/06.
https://doi.org/10.1145/3448300.3468252

1 INTRODUCTION

Apple AirDrop is a file-sharing service that allows users to send photos and other media over a direct Wi-Fi connection from one Apple device to another. As people typically want to share sensitive data exclusively with people they know, AirDrop only shows receiver devices from address book contacts by default. To determine whether the other party is a contact, AirDrop uses a mutual authentication mechanism that compares a user's phone number and email address with entries in the other user's address book [9].

2 VULNERABILITIES

In our paper [5], we discovered two severe privacy leaks in this authentication mechanism. In particular, we showed that it is possible to learn the phone numbers and email addresses of AirDrop users—even as a complete stranger. An attacker only requires a Wi-Fi-capable device[1] and physical proximity to a target.

The discovered problems are rooted in Apple's use of hash functions for "obfuscating" the exchanged contact identifiers, i.e., phone numbers and email addresses, during the discovery process. It is well-known in industry and academia that hashing fails to provide privacy-preserving contact discovery since hash values of phone numbers can be quickly reversed using simple techniques such as brute-force attacks or database lookups [4].

2.1 Sender Leakage

During the AirDrop authentication handshake, the sender always discloses their own hashed contact identifiers as part of an initial discover message. A malicious receiver can therefore learn all hashed contact identifiers of the sender without requiring any prior knowledge of their target.

To obtain these identifiers, an attacker simply can wait (e. g., at a public hot spot) until a target device scans for AirDrop receivers, i. e., the user opens the sharing pane. After collecting the hashed contact identifiers, the attacker can recover phone numbers and email addresses offline. As shown in prior work [4], recovering phone numbers is possible in the order of milliseconds. Recovering email addresses is less trivial but possible via dictionary attacks that check

[1] AirDrop relies on a proprietary Wi-Fi-based link-layer protocol called Apple Wireless Direct Link (AWDL) [8].

Table 1: Structure of the validation record certificate exchanged during the AirDrop authentication handshake. Problematic fields are highlighted in bold.

Field name	Content
Version	2
altDsID	UUID of Apple account
encDsID	UUID of Apple account
SuggestValidDuration	2592000 seconds (= 30 days)
ValidAsOf	date and time
ValidatedEmailHashes	array of SHA-256 hashes
ValidatedPhoneHashes	array of SHA-256 hashes

common email formats such as first.lastname@{gmail.com, yahoo.com, …}. Alternatively, an attacker could utilize data breaches or use an online lookup service for hashed email addresses [2].

This vulnerability was also independently discovered and published by the Apple Bleee project in July 2019 [1], shortly after our initial responsible disclosure to Apple in May 2019.

2.2 Receiver Leakage

AirDrop receivers present their hashed contact identifiers in response to the discover message if they know any of the sender's contact identifiers (e. g., if the receiver has stored the sender's email address). A malicious sender can thus learn all contact identifiers (including the receiver's phone number) without requiring any prior knowledge of the receiver—if the receiver knows the sender.

Importantly, the malicious sender does not have to know the receiver: A popular person within a certain context (e. g., the manager of a company) can exploit this design flaw to learn all (private) contact identifiers of other people who have the popular person in their address book (e. g., employees of the company).

3 PROOF-OF-CONCEPT ATTACKS

We demonstrate two attacks exploiting the vulnerabilities with a proof-of-concept implementation called "AirCollect" that is publicly available on GitHub (cf. §5). It combines the efforts of OpenDrop [7], an open-source AirDrop implementation, with RainbowPhones [3], an open-source rainbow table implementation that is optimized for non-uniform input domains such as mobile phone numbers.

Extracting Phone Number Hashes from Certificates. We extend the OpenDrop [7] implementation with a module that parses the validation record certificates exchanged during the authentication handshake. We depict the structure of validation record certificate in Table 1, which is identical for both sender and receiver devices. Our proof-of-concept code parses the array of phone hashes and passes them to the RainbowPhones tool for lookup.

Rainbow Table: Success Rate vs. Storage. We use the open-source tool RainbowPhones [3] to efficiently find the preimage to a given hashed phone number. Compared to a conventional rainbow table implementation, RainbowPhones features a specialized reduction function that considers the non-uniform input domain of phone numbers [4]. To make RainbowPhones usable for recovering hashes exchanged in the AirDrop protocol, we added support for the SHA-256 hash algorithm.

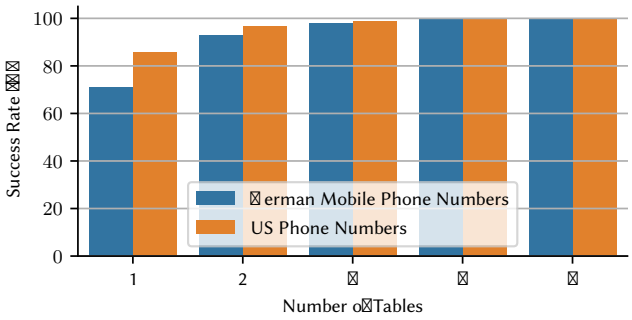


Figure 1: Analysis of success rate for SHA-256 hash reversal with RainbowPhone [3] when combining an increasing number of equally-sized precomputed rainbow tables for German mobile and all US phone numbers.

For this demo, we compute a total 5 tables (each 15.3 MB in size) for German mobile phone numbers that together achieve a success rate of 99.8 % (measured by reversing 10 k hashes of randomly chosen German mobile phone numbers). In our published proof-of-concept implementation, we *omit* these precomputed tables that would allow attackers to reverse almost any given phone number hash in a matter of milliseconds. In Fig. 1, we analyze the trade-off between success rate and storage requirements, finding that even only a single table already achieves 71 % success rate, making the approach attractive for small embedded devices. The figure also includes similar measurements for 5 tables precomputed for all phone numbers in the US that with a total size of 765 MB achieve 100 % success rate (again measured over 10 k randomly chosen samples).

Exploiting Sender Leakage in Practice. Finally, we can execute an attack to exploit sender leakage via a single command-line call on a MacBook.[2] The OpenDrop program starts listening for active AirDrop senders, e. g., iPhones with an open sharing pane, and immediately logs discovered phone numbers to the standard output of the console. An example output would look as follows:

```
$ python3 -m opendrop receive
Announcing service: host opendrop, address fe80::
    c8b9:fbff:fee9:d544, port 8771
Starting HTTPS server
Nearby phone number: +49<...>
```

Our test setup is depicted in Fig. 2.

Exploiting Receiver Leakage in Practice. Similarly, we demonstrate how to exploit receiver leakage, which does not require any interaction with the target. However, we must be able to present a valid AirDrop certificate containing contact identifiers known to the target. For this, we leverage a tool to extract valid AirDrop certificates to be used with AirCollect [6]. An example output after executing the attack looks as follows:

```
$ python3 -m opendrop find
Looking for receivers. Press Ctrl+C to stop ...
Nearby phone number: +49<...>
Found  index 0  ID a019<...>  name John's iPhone
```

[2] Alternatively, we can also use a Linux-based machine such as a Raspberry Pi, but the setup then also requires running an open version of the AWDL protocol [9].

Figure 2: Test setup for demonstrating sender leakage. The iPhone user (right) opens the sharing pane on the device to share a document with the person in the background. The attacker (left) running "AirCollect" on a MacBook immediately sees the sender's phone number.

4 RELATED WORK

The "Apple Bleee" project [1] independently discovered the issue of *sender leakage* and published their findings including a proof-of-concept implementation—two months after our initial disclosure to Apple in May 2019. Their implementation is also based on OpenDrop [7] and uses database lookups to reverse hashes. In contrast, we rely on a custom rainbow table construction that represents an interesting computation/storage trade-off and makes on-the-fly recovery even feasible for small embedded devices. Furthermore, in our demo, we practically show that the recently discovered *receiver leakage* vulnerability can be exploited as well.

5 CONCLUSION

We demonstrated how easy and with how little resources an attacker can learn private information (especially phone numbers) of AirDrop users in proximity. We leave the extension of our prototype to incorporate efficient reversal of email addresses as future work (e. g., by including calls to existing online services [2]).

We responsibly disclosed the issue to Apple in May 2019 as well as a practical solution [5] in October 2020. As of May 2021, Apple has not indicated if they are working on mitigating this issue. This means that Apple users are still vulnerable to the described attacks.

AVAILABILITY

Our proof-of-concept implementation "AirCollect" is available online at https://privatedrop.github.io.

ACKNOWLEDGMENTS

We thank Ann-Kathrin Braun and Daniela Fleckenstein for providing the photograph of our proof-of-concept setup. Also, we thank Christoph Hagen and Christoph Sendner for answering questions about RainbowPhones.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, by the LOEWE initiative (Hesse, Germany) within the emergenCITY center, by the German Federal Ministry of Education and Research and the Hessian State Ministry for Higher Education, Research and the Arts within ATHENE.

REFERENCES

[1] Dmitry Chastuhin. *Apple Bleee: Everyone Knows What Happens on Your iPhone.* July 25, 2019. URL: https://hexway.io/research/apple-bleee/ (visited on 10/15/2020).

[2] Datafinder. *Recover Encrypted Email Addresses.* 2020. URL: https://web.archive.org/web/20191211152224/https://datafinder.com/products/email-recovery (visited on 10/15/2020).

[3] Christoph Hagen and Sebastian Schindler. *RainbowPhones.* 2021. URL: https://github.com/contact-discovery/rt_phone_numbers.

[4] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. "All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers". In: *NDSS.* 2021. URL: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_1C-3_23159_paper.pdf.

[5] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. "PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop". In: *USENIX Security Symposium.* 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/heinrich.

[6] Milan Stute. *Extracting Apple ID Validation Record, Certificate, and Key for AirDrop.* 2020. URL: https://github.com/seemoo-lab/airdrop-keychain-extractor.

[7] Milan Stute and Alexander Heinrich. *OpenDrop: An Open Source AirDrop Implementation.* 2019. URL: https://github.com/seemoo-lab/opendrop.

[8] Milan Stute, David Kreitschmann, and Matthias Hollick. "One Billion Apples' Secret Sauce: Recipe for the Apple Wireless Direct Link Ad hoc Protocol". In: *International Conference on Mobile Computing and Networking.* ACM, 2018. DOI: 10.1145/3241539.3241566.

[9] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. "A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link". In: *USENIX Security Symposium.* 2019. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/stute.

D PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop (USENIX Security'21)

[HHS+21] A. HEINRICH, M. HOLLICK, T. SCHNEIDER, M. STUTE, C. WEINERT. **"PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop"**. In: *30. USENIX Security Symposium (USENIX Security'21)*. Website: https://privatedrop.github.io. Full version: https://ia.cr/2021/481. USENIX Association, 2021, pp. 3577–3594. CORE Rank A*. Appendix D.

https://www.usenix.org/system/files/sec21-heinrich.pdf

PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop

Alexander Heinrich, Matthias Hollick, Thomas Schneider,
Milan Stute, and Christian Weinert, *TU Darmstadt*

https://www.usenix.org/conference/usenixsecurity21/presentation/heinrich

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the
30th USENIX Security Symposium
is sponsored by USENIX.

PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop

Alexander Heinrich Matthias Hollick Thomas Schneider
Milan Stute Christian Weinert

Technical University of Darmstadt, Germany

Abstract

Apple's offline file-sharing service AirDrop is integrated into more than 1.5 billion end-user devices worldwide. We discovered two design flaws in the underlying protocol that allow attackers to learn the phone numbers and email addresses of both sender and receiver devices. As a remediation, we study the applicability of private set intersection (PSI) to mutual authentication, which is similar to contact discovery in mobile messengers. We propose a novel optimized PSI-based protocol called *PrivateDrop* that addresses the specific challenges of offline resource-constrained operation and integrates seamlessly into the current AirDrop protocol stack. Using our native PrivateDrop implementation for iOS and macOS, we experimentally demonstrate that PrivateDrop preserves AirDrop's exemplary user experience with an authentication delay well below one second. We responsibly disclosed our findings to Apple and open-sourced our PrivateDrop implementation.

1 Introduction

Apple AirDrop is a file-sharing service integrated into more than 1.5 billion end-user devices worldwide [5], including iPhone, iPad, and Mac systems, and has been in operation since 2011. AirDrop runs fully offline and only uses a direct Wi-Fi connection in combination with Bluetooth Low Energy (BLE) between two devices. We discovered *two* severe privacy vulnerabilities in the underlying authentication protocol. In particular, the flaws allow an adversary to learn contact identifiers (i.e., phone numbers and email addresses) of nearby AirDrop senders *and* receivers. The flaws originate from the exchange of hash values of such contact identifiers during the discovery process, which can be easily reversed using brute-force or dictionary attacks [35, 42, 66].

Challenge. During authentication, two AirDrop devices run a form of *contact discovery* where they determine if they are mutual contacts, i.e., whether or not they have stored each others' contact information in their address book [92]. A connection is only deemed authentic if the result is positive.

Privacy-preserving contact discovery is commonly addressed via private set intersection (PSI) in the literature (e.g., [55, 59]). PSI protocols, in general, are cryptographic protocols that allow two interacting parties to securely compute the intersection of their respective input sets without leaking any additional data. PSI is already deployed in the real world, e.g., for compromised credential checking in Google's browser Chrome [93] in a business-to-consumer (B2C) context and for calculating ad conversion rates with Google in a business-to-business (B2B) context [51]. In a consumer-to-consumer (C2C) context, PSI has been proposed for preventing cheating in online gaming [20] and most recently for contact tracing in light of the COVID-19 pandemic (e.g., [94]). With our work, we aim to facilitate the deployment of PSI in a C2C context for mutual authentication.

However, the AirDrop scenario poses a unique set of challenges: a solution needs to (a) run completely offline without any third-party server support, (b) consider malicious parties that lie about their address book entries or own contact identifiers, (c) run on mobile devices with restricted energy and computational resources, and (d) preserve the user experience by not adding noticeable authentication delays.

Our contributions. We study the applicability of PSI to realize private mutual authentication for AirDrop. For this, we first systematically explore all possible design options and available building blocks from the literature. Our final solution, called *PrivateDrop*, is based on a Diffie-Hellman-style PSI protocol [53], which is even secure in the presence of malicious actors that actively try to extract sensitive information. We apply a two-way variant of [53] and optimize online performance by minimizing the number of communication rounds and by allowing to precompute expensive operations, e.g., when the device charges overnight. To accommodate malicious inputs, especially attackers lying about their contact identifiers, we propose to use signed PSI inputs [21, 31, 33] that complement AirDrop's current validation records and can be issued using Apple's existing certification infrastructure.

Furthermore, we integrate PrivateDrop into the original AirDrop protocol stack, including the BLE-based discov-

ery mechanism as well as the HTTPS-based authentication phase. We implement both the original AirDrop protocol and our PrivateDrop extension in native code for iOS and macOS, which we open-sourced on GitHub [45].

Finally, in an extensive performance evaluation, we demonstrate that PrivateDrop incurs only negligible overhead in practice. In particular, we experimentally show that the authentication delay stays well below 1 s even for large address books with > 10 k entries, which humans perceive as an "immediate response" [22]. In realistic scenarios, the delay even stays below 500 ms—only a $2\times$ increase compared to the authentication delay in the original insecure AirDrop protocol.

We disclosed both vulnerabilities and our proposed mitigation to the Apple Product Security team and are awaiting their feedback. We summarize our contributions as follows:

(a) We discover and disclose two distinct design flaws in the AirDrop authentication protocol that enable an attacker to learn contact identifiers (phone numbers and email addresses) of nearby devices.
(b) We propose PrivateDrop, a new PSI-based mutual authentication protocol that integrates seamlessly into the current AirDrop protocol stack. Our design is based on a Diffie-Hellman-style PSI protocol [53] and protects against malicious adversaries as well as inputs.
(c) We re-implement the original AirDrop protocol stack, integrate our PSI-based protocol for iOS and macOS, and open-source our code [45].
(d) We experimentally show that PrivateDrop provides immediate responses [22] with < 1 s authentication delay.

Outline. Our paper is structured as follows: We first describe the currently deployed AirDrop protocol (§ 2) and discuss the vulnerabilities we discovered (§ 3). Then, we present our novel PSI-based mutual authentication protocol (§ 4). We furthermore describe our implementation (§ 5), followed by our extensive experimental evaluation (§ 6). Finally, we discuss related work (§ 7) before concluding (§ 8).

2 Background: Apple AirDrop

Apple's file-sharing service AirDrop is integrated in all current iOS and macOS devices. It runs completely offline using a proprietary Wi-Fi link-layer called Apple Wireless Direct Link (AWDL) [90] in combination with Bluetooth Low Energy (BLE). As there exists no official documentation of the involved protocol stack, we describe AirDrop based on the reverse engineering of [92]. In particular, we first define *contact identifiers* and discuss the available *discoverability* settings. Then, we describe the complete technical protocol flow and explain the authentication process as presented in [92].

2.1 Contact Identifiers and the Address Book

Each iOS or macOS device has an address book that is accessible through the *Contacts* application. This address book contains several contact entries that in turn consist of multiple objects such as name or contact information. AirDrop leverages the user's own contact identifiers and their address book entries for authentication purposes. In particular, AirDrop uses phone numbers and email addresses to identify a contact. This is possible as every Apple account (often referred to as *Apple ID* or *iCloud account*) has at least one such contact identifier assigned to it. Apple uses verification emails and SMS to verify the ownership of the email address or phone number, respectively, thus assuring the correctness of the identifiers.

Within the context of this paper, we will only deal with contact identifiers, i.e., phone numbers and email addresses, and disregard the notion of "contacts" that might—in turn—consist of multiple identifiers. We assume there exists a device-local unambiguous mapping for contact identifiers to contact list entries. We use the term *address book (AB)* to refer to the set of contact identifiers of all contact entries in the device's contact list. Note that the AB is controlled by the user and not verified by Apple. In addition, the user's own *contact identifiers (IDs)* are the Apple-verified phone numbers and email addresses that are assigned to the user's Apple account. We use the notation c to refer to an address book entry and ID to refer to an Apple-verified contact identifier.

2.2 Device Discoverability

When opening the sharing pane on an iOS device, nearby devices appear in the user interface if they are discoverable [10]. In particular, receiver devices can be discovered by *everybody* or by *contacts only*, which is the default setting. In either case, an AirDrop sender will attempt to perform a mutual authentication handshake with a responding receiver. Note that the issues addressed in our paper (i.e., the leakage of contact identifiers of sender and receiver during the authentication process) affect both settings.

2.3 Full Protocol Workflow

The AirDrop protocol allows a *sender* to transmit a file or link to a *receiver*. It consists of the three phases *discovery*, *authentication*, and *data transfer*, which we explain based on [92] and depict in Fig. 1: (a) When the sender opens the sharing pane, it starts emitting BLE advertisements that contain a truncated hash for each contact identifier. A receiver compares the sender's hashed contact identifiers with entries in their address book. The receiver activates their AWDL interface if at least one contact match was found in contacts-only mode or if it is discoverable by everyone. The sender then proceeds by searching for AirDrop services with DNS service discovery (DNS-SD) via the AWDL interface. (b) For each discovered service, the sender initiates an authentication procedure via an HTTPS *Discover* request that we detail in § 2.4. If the authentication procedure completes successfully, the receiver's identity is displayed in the sender's user interface. (c) Finally, the sender selects the receiver and sends two subsequent requests: The *Ask* request contains metadata

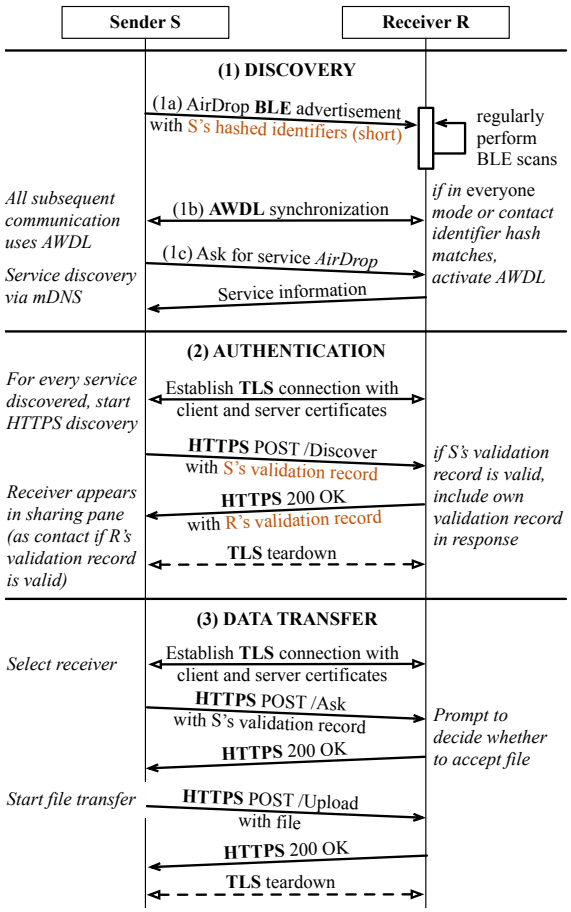


Figure 1: AirDrop protocol (simplified version from [92]). The orange message parts leak the sender's and receiver's contact identifiers, as discussed in § 3.3 and § 3.4, respectively.

about the file, including a thumbnail. The receiver sends their decision on whether to receive the full file. Upon a positive response, the sender continues to transfer the complete file in an *Upload* request or aborts the transaction otherwise.

2.4 Mutual Authentication

An authenticated connection can only be established between users with an Apple ID who are present in each others' address books. In order to authenticate, a device needs to prove that it has registered a certain contact identifier ID_i such as phone number or email address associated with its Apple ID, while the verifying device checks whether ID_i is an address book entry. Authentication involves multiple Apple-signed certificates and a chain of Apple-run certificate authorities (CAs). In particular, AirDrop uses a device-specific certificate σ_{UUID} and a validation record VR_σ, which are both signed by Apple. The devices retrieve them both from Apple once the user logs in to their iCloud account. They can then be used offline in any subsequent AirDrop transaction.

The certificate σ_{UUID} contains an account-specific universally unique identifier (UUID).[1] The certificate is used as a client or server certificate (depending on the role) in the TLS connection. As the UUID in the certificate does not link any contact identifiers, AirDrop uses an Apple-signed *Apple ID validation record* (VR_σ). The validation record contains the UUID from the TLS certificate and all contact identifiers $\mathit{SHA\text{-}256}(ID_1), \ldots, \mathit{SHA\text{-}256}(ID_m)$ that are registered with the user's Apple ID in hashed form. Also, VR_σ includes a signature and the certificate of the signing CA σ_{VR}.[2] Formally, we define VR_σ as follows:

$$VR = (\mathit{UUID}, \mathit{SHA\text{-}256}(ID_1), \ldots, \mathit{SHA\text{-}256}(ID_m)) \quad (1)$$

$$VR_\sigma = (VR, \mathit{sign}(\sigma_{\text{VR}}, VR), \sigma_{\text{VR}}), \quad (2)$$

where $\mathit{sign}(\sigma_{\text{VR}}, VR)$ is the signature of VR for certificate σ_{VR}. During authentication, AirDrop (a) verifies the signature on the received validation record, (b) verifies that the UUID in the certificate matches the one in the validation record, and (c) computes the SHA-256 hash over each normalized[3] address book entry and compares them with the hashes contained in the validation record. Authentication succeeds if all checks pass. If authentication fails on the receiver side, the receiver aborts the connection. However, if authentication fails on the sender side, AirDrop continues the transaction but treats the connection as unauthenticated and the peer as a non-contact. AirDrop shows contacts with their name and picture from the address book in the user interface. Non-contacts are displayed using the device name without a picture instead.

3 Contact Identifier Leakage in AirDrop

We discovered two design flaws in the AirDrop protocol that allow an adversary to learn the contact identifiers (both phone numbers and email addresses) of nearby Apple devices. The two flaws originate from AirDrop's authentication handshake, where hashed contact identifiers are exchanged as part of Apple's validation record. First, we define the threat model and discuss that cryptographic hash functions cannot hide their inputs (called preimages) when the input space is small or predictable, such as for phone numbers or email addresses. Second, we explain where and to what extent AirDrop devices are vulnerable to contact identifier leakage. We responsibly disclosed our findings to Apple (cf. § 8). A subset of the issues presented in the following was independently reported in [25]. However, that report does not address hashed email addresses and receiver leakage (cf. § 3.4), and was published one month *after* our disclosure with Apple. Moreover, there are no signs that [25] followed responsible disclosure.

[1] As an addition to [92], we found that the UUID is not device-specific but equal for all devices using the same Apple account.

[2] We hide the fact that VR_σ contains the complete certificate chain up to Apple's root CA [92] to keep our description short and concise.

[3] Phone numbers are hashed in a normalized digit-only form, e.g., the string "+1 (234) 567-8901" is hashed as "12345678901".

3.1 Threat Model

In this paper, we consider an adversary that wants to learn contact identifiers (phone numbers and email addresses) from non-contact AirDrop devices in proximity. They might then use these identifiers for fraudulent activities such as (spear) phishing attacks or making a profit by selling personal data.

Specifically, the adversary must be in physical proximity of its targets (similar to [88]) and have access to a device with an off-the-shelf Wi-Fi card to communicate via AWDL [89]. We assume that the adversary has full control over the wireless channel and can, e.g., mount machine-in-the-middle attacks [92]. The adversary may lie about its address book (AB) entries and arbitrarily deviate from the protocol description, but cannot break Apple's contact identifier ownership verification (cf. § 2.1), i.e., the adversary is unable to forge valid certificates for arbitrary contact identifiers (IDs).

We assume that Apple is trustworthy as it acts as a certificate authority (cf. § 2.4) and learns the contact identifiers, but not the address book entries, from all of its users through the ownership verification process.

3.2 Recovering Hashed Contact Identifiers

Hashing is insufficient to hide phone numbers or email addresses as the input space is small/predictable [35, 42, 66].

Phone numbers. Recovering the preimage of a hashed phone number can be achieved using brute force because the phone number space is relatively small. For example, a US phone number contains an area code followed by 7 digits. Given this small search space (10^7), it is feasible to check all possible phone numbers on a PC within seconds.

More precisely, a recent work [42] studied three different approaches for efficiently reversing phone number hashes: lookups in large-scale key-value stores, brute-force attacks, and optimized rainbow-table constructions. The authors also modeled a worldwide database of valid mobile phone number prefixes that revealed vast differences in terms of phone number structure between countries and, therefore, the size of the search space (e.g., in Austria, the search space is in the order of 10^{10} compared to 10^7 in the US). Each of the investigated reversal methods was able to reverse SHA-1 hashes with an amortized runtime in the order of milliseconds (e.g., 52 ms for the optimized rainbow-table construction). These results are directly applicable to estimate the effort required for an attacker to recover a phone number from the hashes leaked in AirDrop (cf. § 3.3 and § 3.4). However, since AirDrop uses SHA-256 instead of SHA-1, the runtime and storage requirements stated in [42] likely increase by around factors 3× and 1.6×, respectively [49].

Email addresses. Recovering the preimage of a hashed email address is less trivial but possible via dictionary attacks that check common email formats such as `first.lastname@{gmail.com,yahoo.com,...}`. Alternatively, an attacker could generate an email lookup table from data breaches [48] or use an online lookup service for hashed email addresses [34].

3.3 Contact Identifier Leakage of Sender

During the AirDrop authentication handshake, the sender always discloses their own contact identifiers as part of the initial HTTPS POST /Discover message (cf. Fig. 1). A malicious receiver can therefore learn all (hashed) contact identifiers of the sender *without requiring any prior knowledge* of their target. To obtain these identifiers, an attacker simply needs to wait (e.g., at a public hot spot) until a target device scans for AirDrop receivers, i.e., *the user opens the AirDrop sharing pane*. The target device will freely send a discover message to any AirDrop receiver found during the previous DNS-SD service lookup. Therefore, an attacker can learn the target's validation record without any authentication by simply announcing an AirDrop service via multicast DNS (mDNS). After collecting the validation record, the attacker can recover the hashed contact identifiers offline.

3.4 Contact Identifier Leakage of Receiver

AirDrop receivers present their contact identifiers in the HTTPS 200 OK response to the discover message if they know any of the sender's contact identifiers included in the validation record (cf. Fig. 1). A malicious sender can thus learn all contact identifiers *without requiring any prior knowledge* of the receiver *if the receiver knows the sender*. Importantly, the malicious sender does not have to know the receiver: A popular person within a certain context (e.g., the manager of a company) can exploit this design flaw to learn all contact identifiers of other people who have the popular person in their address book (e.g., employees of the company).

4 PrivateDrop: PSI-based Mutual Authentication for AirDrop

In the following, we describe how PSI can be applied to realize PrivateDrop, our private mutual authentication protocol for AirDrop that protects against both attacks described in § 3.

In general, given sender S and receiver R with verified contact identifiers and size-constrained address books (IDs^S, AB^S) and (IDs^R, AB^R), respectively, a privacy-preserving mutual authentication protocol must ensure that S and R learn *at most* those contact identifiers of the other party that they already have in their address book, i.e., S learns *at most* $AB^S \cap IDs^R$ and R learns *at most* $AB^R \cap IDs^S$.[4]

Private set intersection (PSI) protocols are cryptographic protocols that securely compute the intersection $A \cap B$ for two parties with respective private input sets A and B. For the remainder of this paper, we denote the party obtaining the intersection result as *PSI receiver* and the respective other party as *PSI sender*.[5] Importantly, with PSI, no elements outside the intersection, i.e., from $(A \cup B) \setminus (A \cap B)$, are leaked.

[4] During AirDrop authentication, S learns IDs^R if $IDs^S \cap AB^R \neq \emptyset$ and R learns IDs^S unconditionally, resulting in the vulnerabilites described in § 3.

[5] There also exist PSI protocols where both parties can be receivers, but this property is not required for our authentication purposes.

To instantiate PrivateDrop, we first fix our requirements for the authentication protocol, explore the different design options when applying PSI, choose a suitable PSI protocol from the literature, adapt and optimize it for our use case, and seamlessly integrate it into AirDrop.

4.1 Requirements

Our primary goal is to prevent both attacks described in § 3 by protecting *contact identifiers* (Apple-verified phone numbers and email addresses assigned to a user's Apple account, cf. § 2.1) and *validation records* (Apple-signed lists of hashed contact identifiers, cf. § 2.4). Concretely, in terms of functionality and *privacy* for the AirDrop authentication, we want to simultaneously achieve the following properties:

(a) Disclose validation records only *if both parties are mutual contacts*. If both parties are mutual contacts, they already know at least one contact identifier of the respective other party. Thus, the hash values enclosed in the validation records do not leak personal information via brute-force or dictionary attacks (cf. § 3.2).

(b) In the validation records, disclose only those *contact identifiers that the other party already knows*. Even though mutual contacts already know at least one contact identifier of the respective other party, the validation records contain hash values of *all* registered identifiers. Thus, the hash values of contact identifiers not known to the respective other party leak additional personal information via brute-force or dictionary attacks (cf. § 3.2).

We use *A knows B* as a shorthand for *A has one of B's verified contact identifiers (IDs^B) in their size-constrained (cf. § 4.5) address book (AB^A)*, or formally: $AB^A \cap IDs^B \neq \emptyset$.

In terms of *performance*, we want to minimize computation as well as communication overhead. This is important to achieve a low energy consumption for battery-driven mobile devices and to deliver a great user experience with immediate responses. Since AirDrop is primarily used on mobile devices, which might be offline from time to time, our solution must be fully *decentralized* and cannot involve an external server. Furthermore, we have to consider that parties might act *maliciously*, i.e., may try to apply arbitrary strategies with the intent to extract personal information.

4.2 Design Options and Final Design

We now describe how to apply PSI to realize private mutual authentication for AirDrop, considering the requirements defined in § 4.1. The main task is to replace the insecure exchange of hash values that happens in the original authentication phase as a result of sending validation records (cf. § 2.4).

Our high-level idea summarized in Fig. 2 is to have two consecutive PSI executions. The first execution ensures the AirDrop sender knows the receiver, the second that the AirDrop receiver knows the sender. Afterward, as each party is assured that it is stored in the respective other party's address book, it is safe for them to reveal their contact identifiers and validation records. In the following, we detail how to configure the PSI executions to achieve the described outcome by systematically analyzing all possible design options.

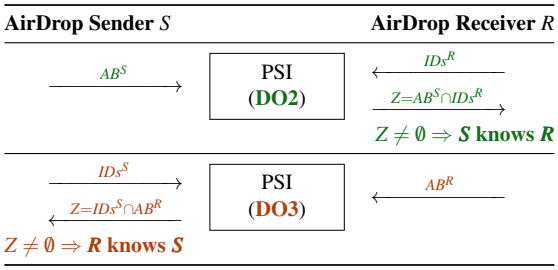


Figure 2: PrivateDrop's PSI-based mutual authentication protocol for AirDrop. The PSI protocols are instantiated using DO2 (green) and DO3 (orange), cf. § 4.2. Inputs are the parties' contact identifiers (IDs) and address books (AB).

Table 1: Available design options (DO) to use PSI for private mutual authentication in AirDrop. Possible inputs are contact identifiers (IDs) and address books (AB). The parties can act as PSI sender (PSI S) or PSI receiver (PSI R).

Design Option	DO1	DO2	DO3	DO4
Role of AirDrop Sender	PSI S	PSI S	PSI R	PSI R
Input of AirDrop Sender	IDs	AB	IDs	AB
Role of AirDrop Receiver	PSI R	PSI R	PSI S	PSI S
Input of AirDrop Receiver	AB	IDs	AB	IDs

The design options (DOs) listed in Tab. 1 differ in (a) the PSI inputs for the AirDrop sender and receiver, i.e., contact identifiers and address books, (b) the roles the parties take in PSI, and (c) the order in which the DOs are executed.

Note that we exclude combinations where both parties input their contact identifiers since the intersection will always be empty. Likewise, we do not consider both parties using their address book as input, since this variant (formalized in [32] as private contact discovery between two users) yields the parties' common contacts (i.e., finds "friends of friends" [12]) but does not determine whether they are mutual contacts.

Regarding the assignment of the PSI roles and the execution order, we can exclude further combinations. As both AirDrop sender and receiver must be assured of being mutual contacts, each must act as PSI receiver once. In the authentication process, the AirDrop sender should be the first to reveal information as otherwise malicious senders could easily extract such information from a large number of innocent receivers by triggering the authentication process. Therefore, the options must be chained such that the AirDrop receiver acts as PSI receiver first (DO1 or DO2) and as sender second (DO3 or DO4). In the following, we discuss the two remaining possibilities.

DO1 → DO4. Here, the PSI sender has their contact identifiers as input, whereas the PSI receiver has their address book as input. As a result, each party is assured that the other party is one of its contacts. This is the exact semantic as

in the original (insecure) authentication protocol. However, since malicious AirDrop receivers do not necessarily abort after receiving an empty result set in the first PSI execution, AirDrop senders have no proof that the receivers know them before revealing their contact identifiers. Since we strictly want to avoid this information leakage (cf. § 3.3), we discard DO1 → DO4.

DO2 → DO3. Here, the PSI sender has their address book as input, whereas the PSI receiver has their contact identifiers as input. At the end of the authentication process, each party can be assured that it is stored in the respective other party's address book. Thus, the AirDrop sender can safely share their contact identifiers that appeared in the outcome of the DO3 execution since the other party already has them stored.

In conclusion, by executing DO2 → DO3 in that particular order (as visualized in Fig. 2), we can fulfill the functional and privacy requirements defined in § 4.1, and prevent our attacks described in § 3.

4.3 Choice of PSI Protocol

Now that we fixed in which order two PSI protocols have to be run, we need to find instantiations. In the literature, many two-party PSI protocols are proposed that could be applied (cf. § 7.2). Especially, a sub-category of PSI protocols specializes in *unbalanced* set sizes, where one party has a much larger input set than the other [26, 27, 55, 59, 82]. The protocols [26, 27] are based on homomorphic encryption with communication linear in the size of the smaller set, but they are computationally expensive. The fastest unbalanced PSI protocols for mobile clients [55, 59, 82] shift most public-key operations to an input-independent precomputation phase and send an encrypted and compressed representation of the larger input set ahead of time to achieve fast online runtimes. Moreover, the protocols of [55] provide security against malicious PSI receivers but only work for semi-honest senders.

However, even though we deal with unbalanced sets, here, the size of the larger input set is determined by the maximum number of address book entries. The size of address books can be reasonably assumed to be well below 100 k and is not in the order of hundreds of millions as considered for unbalanced PSI. Thus, protocols based entirely on public-key encryption (which are extremely inefficient at a large scale) can achieve practical performance. In our setting, *both* parties are not constrained by business incentives or severe legal consequences to behave semi-honestly. Therefore, we must choose a protocol with security against a malicious sender *and* receiver. Furthermore, AirDrop is a protocol that is performed ad-hoc with random communication partners such that distributing encrypted databases in advance is not possible. Finally, we aim at providing industry-grade implementations for integration into Apple's ecosystem. Therefore, we need a simple protocol that does not require complex libraries for oblivious transfer or garbled circuits as needed in the most efficient protocols of [55, 59].

AirDrop/PSI Sender S		AirDrop/PSI Receiver R
Input: $AB = \{c_1, \ldots, c_n\}$		Input: $IDs = \{ID_1, \ldots, ID_m\}$
Output: $\perp$		Output: $AB \cap IDs$
$k, r \xleftarrow{\$} \mathbb{Z}_q$		For $i = 1$ to m:
		$h_i = H(ID_i)$, $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$,
		$y_i = h_i^{\alpha_i}$
For $i = 1$ to m:	$\xleftarrow{(y_1,\ldots,y_m)}$	
$z_i = y_i^k$, $a_i = y_i^r$		
$c = H(y_1, z_1, a_1, \ldots, y_m, z_m, a_m)$		
$p = r + k \cdot c$		
	$\xrightarrow{(z_1,\ldots,z_m)}$	
	$\xrightarrow{(a_1,\ldots,a_m),p}$	$c = H(y_1, z_1, a_1, \ldots, y_m, z_m, a_m)$
For $j = 1$ to n:		For $i = 1$ to m:
$u_j = H(H(c_j), H(c_j)^k)$		Abort if $y_i^p \neq a_i \cdot z_i^c$
		$v_i = H(h_i, z_i^{1/\alpha_i})$
	$\xrightarrow{\{u_1,\ldots,u_n\}}$	output $\{ID_i \in IDs \mid \exists j : u_j = v_i\}$

Figure 3: Maliciously secure PSI protocol of [53] applied to DO2 (cf. § 4.2). The non-interactive zero-knowledge AND-proof of knowledge is marked in blue [16, 37, 85].

The PSI Protocol of [53]. Considering all requirements, we resort to a public key-based PSI protocol proposed by Jarecki and Liu [53]. This Diffie-Hellman-style protocol extends the work of Baldi et al. [13] by adding malicious security via zero-knowledge proofs.[6] The required public key operations can be efficiently instantiated with elliptic curve cryptography, for which there exist industry-grade libraries such as MIRACL [68] and built-in operating system capabilities (Apple CryptoKit [7] in iOS and macOS).

In Fig. 3, we summarize the PSI protocol of [53] applied to our use case. Specifically, we show the application to DO2 (cf. § 4.2). The application to DO3 works analogously with the same type of inputs (address book AB for PSI sender, identifiers IDs for PSI receiver), but the assignment of AirDrop sender/receiver to PSI sender/receiver is swapped.

For simplicity, H in our description denotes a hash function that maps either one or multiple bit strings or group elements to a short bit string of fixed length or an element in a multiplicative group of prime order q. The respective input and output domains are clear from the context. We instantiate H with the SHA-2 family [69] in our implementation (cf. § 5.2).

Informally, the protocol works as follows: (a) the PSI receiver hashes its input elements ID_i with a collision-resistant hash function H to group elements, encrypts the hash values h_i with random keys α_i, and sends the resulting values y_i to the PSI sender; (b) the PSI sender additionally encrypts the received elements with a random secret key k and sends the results z_i to the receiver; (c) the PSI receiver "removes" its own keys α_i such that it obliviously obtains the encryption of its inputs under the sender's key k; and finally (d) the PSI sender sends hashed encryptions u_j of its own input elements c_j in random order to the receiver, who then can compare the

[6] More precisely, malicious security is proven for an *adaptive* PSI functionality, where the receiver makes a series of adaptive queries instead of inputting its set as a whole. However, as the authors argue, any efficient adversary is committed to all its inputs at the execution time, and thus the adaptive functionality can be assumed to be equivalent to regular PSI [13].

values to determine the intersection. Following the PSI protocol of [76], the bitlength l of the values u_j can be reduced to $\lambda + 2\log_2(n)$, where λ is the statistical security parameter (which we set to $\lambda = 40$ in our implementation), and n is an upper bound on the number of address book entries each party has. This yields negligible failure probability $2^{-\lambda}$.

To achieve malicious security, the protocol utilizes a zero-knowledge proof of knowledge that makes sure the PSI sender knows and uses the same key k for computing all values z_i. This requires a so-called AND proof over the individual exponentiations. For an efficient and straight-forward instantiation, we choose Schnorr's DLOG proof [85] and apply the Fiat-Shamir heuristic [16, 37] to turn it into a non-interactive version (in the random oracle model), which does not require additional communication rounds (cf. blue part in Fig. 3).

The protocol in Fig. 3 leaks some information via the number of inputs. For example, one can learn whether an AirDrop sender is popular from the number of address book entries. To prevent such leakage, we pad the input sets with dummy elements to a globally fixed upper bound. For example, it is reasonable to limit the number of address book entries to $n = 10\text{k}$ and the number of contact identifiers to $m = 10$. In § 6, we assess the practical performance implications of such limits by conducting experiments with variable m and n.

4.4 Optimizing PSI for PrivateDrop

When integrating the PSI protocol of Fig. 3 into AirDrop, we apply several performance improvements.

Precomputation. First, it is possible for the PSI sender to generate the key k and compute the values u_i ahead of time. This can be done, e.g., overnight when the device is charging. It is only necessary to update the precomputed values as address book entries change. Since AB is the bigger input set, this removes the largest computation bottleneck from the protocol execution. Likewise, the PSI receiver can precompute the values y_i, which change seldomly. Similar precomputation techniques were proposed for passively secure DH-style PSI in [59, 82], and with security against malicious clients in [55]. The security of our protocol follows from the security of the protocol of [53]. Concretely, the simulation-based proof of [53] applies equally, as the parties' views remain identical.

Reusage. Moreover, it is possible to reuse the precomputed values across sessions. In previous works [55, 59, 82] that consider large-scale databases as input sets, the precomputed values are reused by encoding and distributing them in probabilistic data structures like Bloom or Cuckoo filters against which OPRF evaluations are checked.

From a standalone perspective, this allows for user tracking, but in AirDrop, users can already be tracked via the UUID in the TLS certificate used for establishing the protocol communication channel (cf. § 2.4). Avoiding user tracking in the entire AirDrop execution is an important area for future work. However, reusing precomputed encryptions of address book entries over longer periods of time allows tracking changes in the contact composition, i.e., how many contacts were added or removed since the last protocol execution. Even if no changes occur, this leaks some information, e.g., no new person was met or no person was "unfriended". In case this leakage should be avoided, fresh encryptions should be precomputed and never be reused.

Round Complexity. In terms of round complexity, it is possible to bundle the last two messages from the PSI sender to the receiver without changing the receiver's view. Thus, the PSI protocol consists of only one round, and the PSI receiver may ignore the received values u_i in case the zero-knowledge proof verification fails.

Furthermore, we optimize the sequential yet independent execution of DO2 and DO3. For this, we bundle the second message of DO2 with the first message of DO3. In total, both protocol executions require sending three messages, thus two rounds. Importantly, directly including the first DO3 message in the last DO2 message does not negatively impact the AirDrop sender in case of engaging with a malicious receiver. This is because in a sequential execution, the AirDrop sender gets no response at the end of DO2. Also, a malicious AirDrop receiver cannot learn any additional private information from receiving encryptions of hashed contact identifiers. Moreover, since the AirDrop receiver gets no response at the end of DO3 and the sender's inputs can be verified (cf. § 4.5), malicious behavior exploiting the sequential execution of the online phases can only influence correctness, but not input privacy.

Note that instead of our proposed three message protocol, it would be possible to further parallelize computation with a fully symmetric execution of DO2 and DO3. This would require sending four messages but can still be done in two rounds. However, to prevent malicious senders from causing unnecessary work for innocent receivers (denial-of-service attacks), we require the sender to first process the receiver's inputs and reveal its encrypted address book entries before starting the computation (cf. § 4.2). Moreover, the potential gain in overall efficiency via additional parallelization is negligible, since the constant overhead caused by one communication round ($\approx 100\,\text{ms}$, cf. Fig. 8) is larger than the entire online computation ($< 50\,\text{ms}$ even for $m = 10$ IDs, cf. Fig. 7).

4.5 Countering Privacy Attacks

The security properties of the PSI protocol in Fig. 3 prevent malicious parties from learning private information even when arbitrarily deviating from the protocol definition. However, malicious parties might tamper with the protocol inputs, which cannot be prevented by the protocol itself since this is an attack on the ideal functionality of set intersection. We now discuss the impact of such attacks and how to counter them by leveraging Apple's existing certification infrastructure.

Malicious Sender. A malicious AirDrop sender could try to obtain sensitive contact information of, e.g., VIPs by including a VIP's publicly known email address in their address

book. The PSI protocol then yields a match, and the vulnerable hash values of all contact identifiers of the VIP are sent in subsequent steps of the AirDrop protocol (including, e.g., the hashed phone number).

To prevent this attack, we modify the AirDrop protocol flow to release only hashed contact identifiers (in the validation record) for which a match in the PSI protocol was found. This requires a change to the current AirDrop validation record, which contains all contact identifiers, cf. Eqs. (1) and (2) on p. 3. In particular, we create individual validation records for each of the user's m contact identifiers ID_i as follows:

$$VR_i = (UUID, SHA\text{-}256(ID_i)), \quad \forall i \in 1,\ldots,m \tag{3}$$

$$VR_{\sigma,i} = (VR_i, sign(\sigma_{VR}, VR_i), \sigma_{VR}). \tag{4}$$

This yields a scalable solution as creating and distributing the validation records is a one-time cost, and the number of IDs per user m is expected to be small (e.g., $m = 10$).

Malicious Receiver. A malicious AirDrop receiver who knows the sender could try to trick the sender into believing they are mutual contacts by using contact identifiers that are stored in the sender's address book with high probability (e.g., emergency phone numbers). Moreover, with the same approach, a malicious AirDrop receiver can test whether the sender knows a specific person. To prevent such attacks, we propose to have the encrypted contact identifiers signed by Apple. The resulting protocol is then closely related to authorized PSI (APSI) [31, 33] and PSI with certified sets [21].

Similarly to the individual validation records in Eq. (4), we introduce Apple-signed certificates that contain the UUID and the precomputed values y_i for the user's contact identifiers:

$$Y_i = (UUID, y_i), \quad \forall i \in 1,\ldots,m \tag{5}$$

$$Y_{\sigma,i} = (Y_i, sign(\sigma_{VR}, Y_i), \sigma_{VR}) \quad . \tag{6}$$

PrivateDrop verifies that the UUID in Eq. (5) equals the one in the TLS certificate to prevent reuse by another party, thus, mitigating replay and machine-in-the-middle attacks. As with Eq. (4), this is a lightweight addition that does not require major changes in the existing infrastructure. The keys α_i can still be chosen on the client device. Only a simple zero-knowledge protocol must be run with Apple to make sure y_i is actually an encryption of a legitimately hashed contact identifier and the client device is in possession of the keys α_i. This can again be efficiently instantiated with Schnorr's protocol [85] and the Fiat-Shamir heuristic [16, 37] (cf. § 4.3). Alternatively, Apple could choose the keys α_i and hand them to client devices together with signed values $Y_{\sigma,i}$.

Brute-force. Finally, either party could try to guess contact identifiers of the other party by adding a large number of "fake" address book entries (so-called enumeration attacks [42]). However, in contrast to offline brute-force attacks, where up to millions of guesses can be checked per second, the success probability is significantly lower since we strictly limit the size of the input sets to a reasonable upper bound (e.g., $m = 10$ and $n = 10\,\text{k}$, cf. § 4.3).

Table 2: Overhead of PrivateDrop's PSI-based mutual authentication protocol on n address book entries and m contact identifiers, respectively. $|q|$ is the size of group elements, $|sign|$ the size of signatures on encrypted contact identifiers, and l the length of hashes u_i.

Phase	Precomputation		Online	
Computation	Sender S	Receiver R	Sender S	Receiver R
Exp.	$m_S + n_S$	$m_R + n_R$	$3m_S + 3m_R$	$3m_S + 3m_R$
Hash calc.	$m_S + 3n_S$	$m_R + 3n_R$	$2m_S + m_R + 2$	$m_S + 2m_R + 2$
Communication	0		$(3m_S + 3m_R + 2) \cdot \lvert q\rvert + (n_S + n_R) \cdot l + (m_S + m_S) \cdot \lvert sign\rvert$	

4.6 Our PrivateDrop Protocol

In Fig. 4, we show our full PSI-based mutual authentication protocol for AirDrop. Its computation and communication overhead is summarized in Tab. 2. For the computation overhead, we count the required exponentiations and hash operations. We assume that verifying each signature requires one such exponentiation and one hash operation. Obtaining the signature on the values y_i is ignored since the exact overhead depends on the chosen implementation. In case Apple provides keys α_i along with values Y_i and signatures $sign(\sigma_{VR}, Y_i)$, the additional communication overhead in the precomputation phase is only $O(m)$. Otherwise, if the keys are chosen on the client device, a non-interactive zero-knowledge Schnorr proof requires additional computation with $O(m)$ exponentiations and hash operations.

Overhead. Overall, in the precomputation phase, both parties have a computation overhead of $O(m+n)$, which is a one-time cost. In the online phase, the computation overhead is $O(m)$, with $m \ll n$, while the communication overhead is $O(m+n)$. Due to n still being fairly limited in practice (e.g., $n = 10\,\text{k}$) and the availability of a low-latency and high-bandwidth Wi-Fi connection, this communication overhead is very well manageable (cf. our experiments in § 6).

5 Implementation and Integration

We implement both the original AirDrop protocol and our PrivateDrop extension for iOS and macOS to empirically study the overhead caused by PSI. We do not use Apple's closed source AirDrop implementation to provide a fair comparison between non-PSI and PSI. In the following, we discuss our implementation (including mDNS and HTTPS communication) and our integration of PrivateDrop into the original AirDrop protocol stack. Our open-source implementation is available on GitHub [45].

5.1 Implementation of the Base Protocol

Apple does not expose or document a low-level AirDrop API that would allow us to integrate our PrivateDrop extension and conduct a fine-grained performance evaluation. Using an existing open-source implementation of AirDrop [46] is also not an option as it is written in Python, which is not supported on iOS and not optimized for performance.

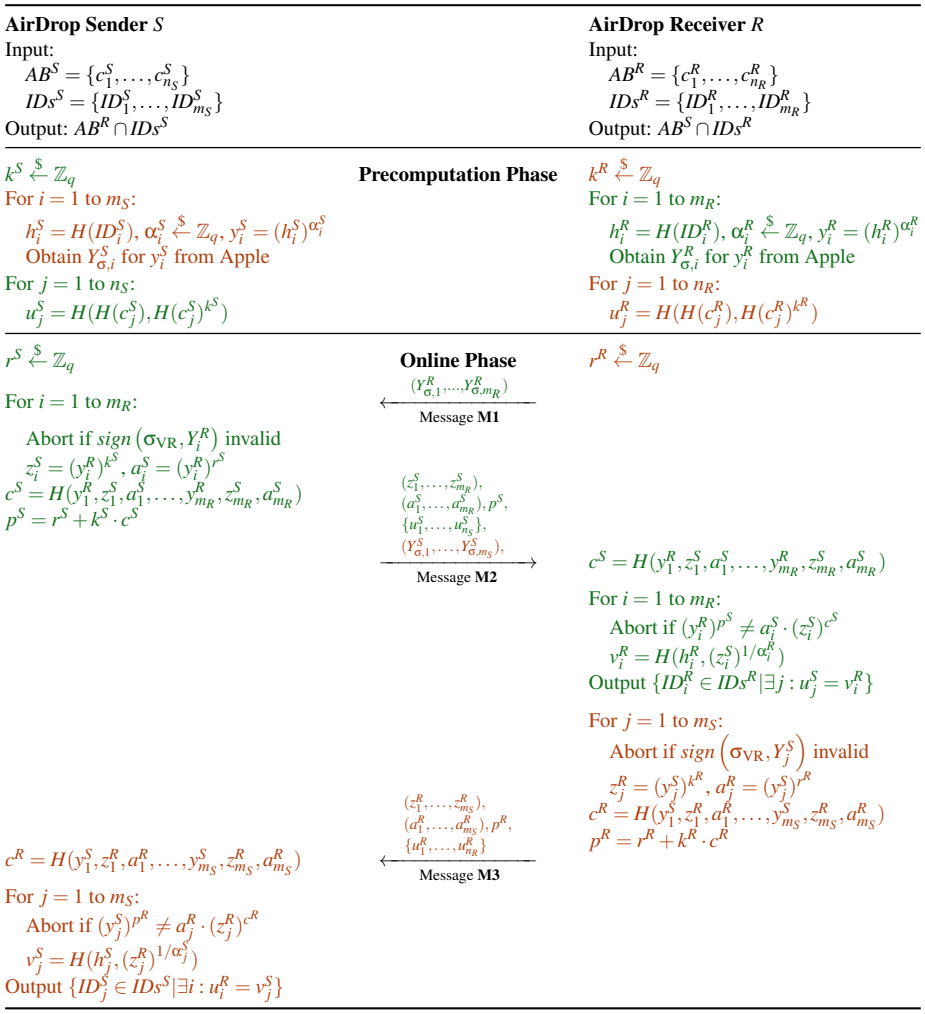


Figure 4: PrivateDrop's full PSI-based mutual authentication protocol for AirDrop. The protocol is based on the optimized and interleaved execution of DO2 (green) and DO3 (orange), cf. Tab. 1 and Fig. 2, divided into a reusable precomputation and an online phase.

Therefore, we re-implement the full AirDrop protocol stack in Swift, Apple's modern programming language that compiles down to assembler code. In particular, we use Apple's public NetService API [8] to announce services via mDNS and bootstrap communication over the AWDL interface. In addition, we use SwiftNIO [9] to achieve high-performance asynchronous network operations and to implement HTTPS communication. In App. C, we show that our AirDrop implementation performs very similar to Apple's.

AirDrop's validation records are implemented using cryptographic message syntax (CMS) [47]. To provide the best integration with Apple's existing certification infrastructure, we also implement the signatures $Y_{\sigma,i}$ in Eq. (6) in CMS. For validation, we use the OpenSSL library [71], as Apple's Security framework provides CMS support only on macOS but not on iOS [6]. The individual validation records $VR_{\sigma,i}$ in Eq. (4) are not part of our implementation.

5.2 Implementation of the PSI Operations

Implementing our PSI protocol requires access to low-level elliptic curve (EC) operations, for which we would have liked to utilize built-in operating system capabilities. Unfortunately, Apple's Swift-based CryptoKit [7] does not expose the required point operations, e.g., addition and scalar multiplication. As an alternative, we use the established open-source library Relic [11]. Compared to other third-party candidates such as MIRACL [68] or libecc [15], Relic is focused on efficiency [73, 81] and portability with support for all relevant architectures, i.e., arm64 (iOS and macOS) and x86_64 (macOS). Also, Relic is written in C, which integrates well with our Swift-based protocol implementation.

We instantiate all primitives to provide a security level of 128 bit. Our Diffie-Hellman-based PSI implementation uses the standardized elliptic curve P-256.

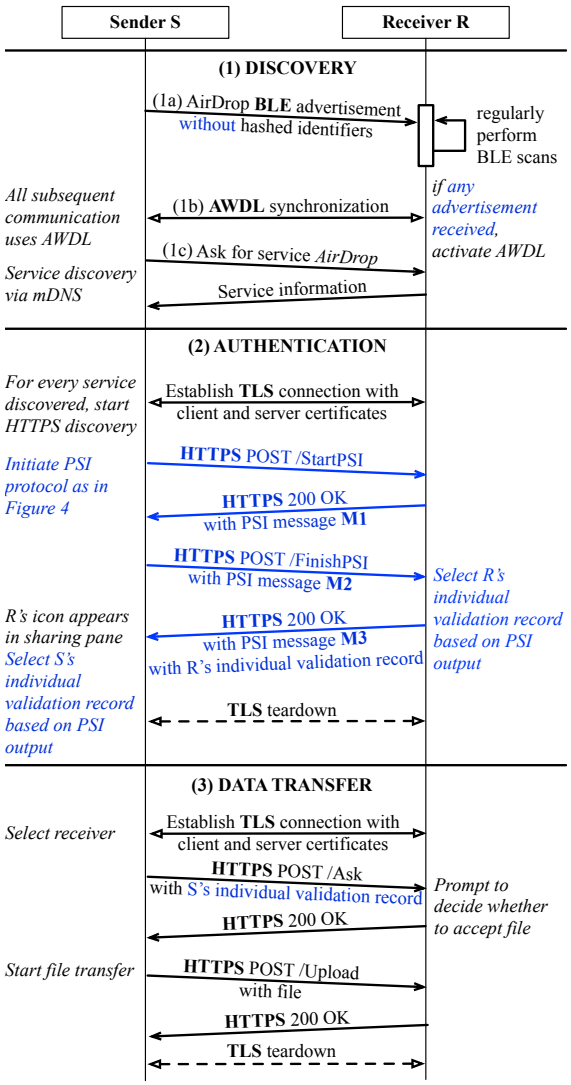


Figure 5: PrivateDrop protocol; changes to the original AirDrop protocol (cf. Fig. 1) highlighted in blue.

5.3 Integration with the HTTPS Handshake

In order to integrate PrivateDrop into AirDrop's HTTPS protocol, we introduce two new HTTPS messages into the authentication phase that we depict in Fig. 5. In particular, we introduce *StartPSI* and *FinishPSI* that include the three messages M1, M2, and M3 from our optimized PSI protocol (cf. Fig. 4) as payload. The protocol is performed immediately after the mDNS discovery is completed and replaces the original HTTPS *Discover* exchange. Since the AirDrop sender acts as the HTTPS client in the protocol, the initial HTTPS request contains no payload and simply signals the receiver to initiate the PSI protocol.

Selecting Individual Validation Records. The output of the PSI protocol determines which individual validation records $VR_{\sigma,i}$ are included in the follow-up requests. If the PSI protocol yields no matches, no validation records are included. If the PSI protocol yields one or more matches, one randomly chosen individual validation record that corresponds to one of the matches is included in the request. Note that, in principle, we could include the validation records for all matches. However, this would yield no benefit as one contact identifier is sufficient to uniquely identify the other party based on the user's address book.[7] On the contrary, transmitting multiple validation records would increase communication overhead and require the receiver to verify multiple signatures.

Communication Rounds. Note that after processing M2, the receiver has already selected the appropriate individual validation record and can send it back to the sender with M3. The sender will include its individual validation record in the *Ask* request when initiating a file transfer. By piggybacking the receiver's validation record to M3, we avoid one additional communication round that would be necessary to exchange $VR_{\sigma,i}$ after the PSI protocol has completed. In total, our PSI-based protocol only incurs one additional communication round compared to the original authentication.

5.4 Integration with the BLE Advertisements

AirDrop's BLE advertisements contain the first two bytes of the sender's hashed contact identifiers, which are also part of the validation record. Receivers use these hashes to check if the sender is a potential contact match and whether they should turn on their AWDL interface to conduct the full authentication handshake. As shown in [92], this mechanism provides no additional security as it can easily be circumvented with brute force. Therefore, the short hashes appear to be an optimization to prevent wakeups of the receiver's Wi-Fi radio that unnecessarily drain the device's battery.

As the purpose of our work is to prevent any leakage of personal information, we propose to not include any (even shortened) contact identifiers and simply set the fields to a fixed value, e.g., `0x0000`. Then, whenever AirDrop receivers overhear such an advertisement, they activate their AWDL interface unconditionally. Coincidentally, this behavior is already implemented by AirDrop receivers that are discoverable by everyone (cf. § 2.2), so we do not expect that this change will incur any practical hurdles.

5.5 Towards Replacing AirDrop

We implemented a fully-functional PrivateDrop prototype. The following changes have to be made by Apple for turning PrivateDrop into a drop-in replacement for AirDrop, which can be deployed with iOS and macOS updates, and

[7] We assume an unambiguous mapping of contact identifiers to contact entries in a user's address book. If a user assigned the same identifier to multiple contacts, then having multiple validation records could help to resolve the ambiguity. In any case, if AirDrop is unable to uniquely identify the other party, it should inform the user, e.g., by displaying an appropriate message. Note that Apple validates ownership of contact identifiers via verification emails or SMS (cf. § 2.1), which prevents multiple registrations, e.g., when users share an office phone number.

Table 3: Experiment parameters.

Protocols	AirDrop, PrivateDrop	
Set sizes	# identifiers m	1, 10, 20
	# address book entries n	100, 1 000, 5 000, 10 000, 15 000
Hardware	Sender (macOS 11) Receiver (iOS 14)	MacBook Pro 15" 2019 iPhone 12 mini
Network connection	Apple Wireless Direct Link (AWDL) [90], USB cable	

requires no hardware modifications: (a) To ensure limited backward compatibility with the orignal AirDrop protocol, PrivateDrop-enabled devices should support AirDrop's *Discover* request but never include AirDrop's validation record to protect themselves against identifier leakage (cf. § 3). PrivateDrop devices would then always appear as non-contacts to AirDrop devices. Note that *downgrade* attacks, i.e., forcing two PrivateDrop devices to use the legacy AirDrop protocol, will hence merely result in unauthenticated connections as PrivateDrop devices will never exchange their validation records with AirDrop devices. (b) Apple's CA infrastructure must be extended to issue $VR_{\sigma,i}$ and $Y_{\sigma,i}$ values. (c) PrivateDrop should use the system's Contact API to provide input for the contact discovery. For evaluation purposes, we use randomly generated contacts. (d) Our implementation currently does not integrate BLE discovery, because iOS hides Apple-specific advertisements in the scan responses and prohibits emitting them for third-party applications. (e) Finally, PrivateDrop currently does not implement individual validation records but uses the AirDrop validation records VR_σ to match the Apple-signed TLS certificates.

6 Experimental Evaluation

We evaluate the performance of PrivateDrop based on our implementation for AirDrop (cf. § 5). To this end, we conduct an extensive experimental evaluation using different Apple devices and variable input sizes over the devices' AWDL interface. We show that the median discovery delay is *well below one second* in any practical setting. In the following, we explain our evaluation metrics and experimental setup. We then present and discuss the evaluation results.

6.1 Evaluation Metrics

We assess the protocol's performance in terms of runtime or *delay*. In particular, we time the protocol flow at several reference points to measure (a) computational overhead, i.e., time spent for calculating cryptographic operations, (b) network overhead, i.e., time spent for transmitting data over the data channel, and (c) overall runtime, i.e., time spent for executing the complete discovery process.

6.2 Experimental Setup

We conduct all experiments using our PrivateDrop and AirDrop implementations (cf. § 5) and summarize all other experiment parameters such as set sizes, hardware, and network environments in Tab. 3.

Set Sizes. Our complexity analysis in § 4.6 shows that the online PSI overhead depends on the number of identifiers m and address book entries n. A previous online study found that Apple users have $n = 136$ contacts on average [92]. Therefore, we select values for n in this order of magnitude but also include values up to $n = 15\,000$ to assess potential scalability limits. Similarly, we select m to cover moderate and extreme limits (1 to 20). For simplicity of presentation, the input sizes are the same for both sender and receiver in all our experiments, i.e., $m = m_S = m_R$ and $n = n_S = n_R$.

Hardware and Network Connection. We use up-to-date Apple devices for the evaluation, in particular, an iPhone 12 mini and a MacBook Pro (2019). A mix of iOS and macOS devices allows us to conduct experiments via a cable network connection (USB) in addition to AWDL, thereby measuring the impact of network-induced delays. In all experiments, the MacBook acts as the sender and the iPhone as the receiver to ensure comparable results.

Environment. We conduct all experiments in a home office environment,[8] where we cannot control interfering Bluetooth and Wi-Fi transmissions. This interference might contribute to the high variance of our AWDL experiments (cf. Fig. 9), which was not observed in previous experiments that used a Faraday tent [90]. We run cable-based experiments to isolate the impact of PrivateDrop, while the AWDL experiments help us to understand performance under real-world conditions.

Test Suite. We implemented a benchmark application for iOS and macOS based on PrivateDrop (cf. § 5) that allows us to define a *scenario*. A scenario is comprised of a fixed set of experimental parameters such as the set sizes and the choice of sender and receiver devices (cf. Tab. 3). For each scenario, we run 100 experiments (Monte Carlo) that each consist of a complete protocol execution. To avoid systematic errors introduced by temporal disturbances, we schedule the individual runs for each scenario in a round-robin fashion. The bar plots indicate the median delay over all runs, and the error bars indicate the 0.05 and 0.95 quantiles. Unless otherwise stated, we measure the delays on the sender side. Each experiment consists of a full protocol run as well as a preparation and cleanup phase: (a) Preparation: we generate the address book at random, precompute the values u_i, and wait until both sender and receiver are ready. (b) Execution: we run a complete protocol execution starting from the DNS-SD discovery to the upload of a file. (c) Cleanup: we shut down the HTTPS and DNS-SD server to close all connections.

[8] Our institution mandated home office due to the COVID-19 pandemic.

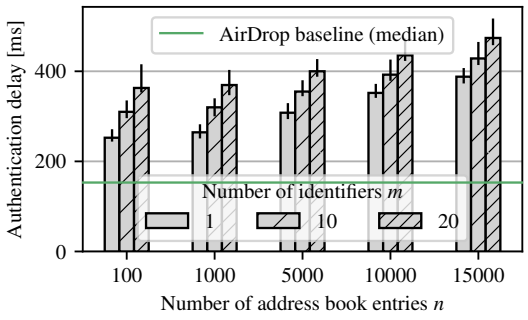


Figure 6: Overall authentication delay for AirDrop (baseline) and PrivateDrop with different set sizes (m,n).

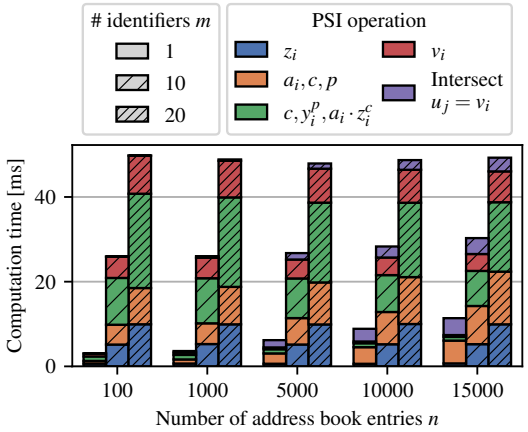


Figure 7: Computation time for the PSI operations on an iPhone 12 with different set sizes (m,n).

6.3 Authentication Delay

We first empirically measure the performance of PrivateDrop's online phase for variable set sizes n and m (cf. Tab. 3). For this, we run a set of experiments between the MacBook Pro 2019 (sender) and iPhone 12 (receiver). In order to minimize noise introduced by the wireless channel, we conduct this experiment via a USB cable connection between sender and receiver. We later evaluate the impact of the wireless channel in § 6.4.

Overall Delay. In Fig. 6, we show the delay of the complete authentication phase (phase (2) in Figs. 1 and 5), for PrivateDrop and AirDrop. AirDrop authentication is independent of m and n, and, therefore, we include the median delay as a baseline. In contrast, the PrivateDrop runtime increases with both m and n as expected. Our results for PrivateDrop show that for moderate settings ($m = 10$, $n = 1\,000$), the median authentication delay is increased by $2\times$ compared to AirDrop. Even for extreme scenarios ($m = 20$, $n = 15\,000$), the overall delay stays below 500 ms. This satisfies our user experience requirement as humans perceive any delay below 1 000 ms as an "immediate response" [22].

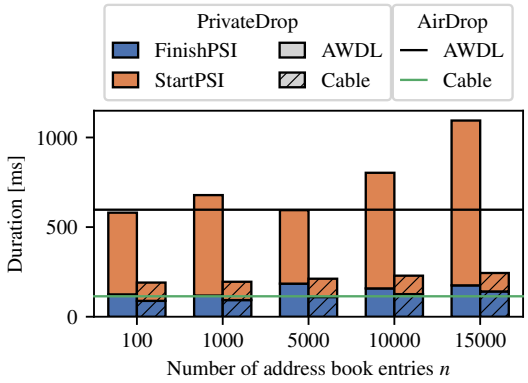


Figure 8: Transmission delay of AWDL and cable connections for the AirDrop (*Discover*) and PrivateDrop (*StartPSI* and *FinishPSI*) requests for a fixed number of identifiers $m = 10$.

PSI Delay. We closer investigate the impact of the PSI online phase on the overall authentication delay. Fig. 7 shows the computation time of the individual operations on an iPhone 12. In fact, only computing the actual intersection depends on the number of address book entries n (cf. violet parts in Fig. 7) and is at most 5 % of the total time for $n = 15\,000$. All other arithmetic operations increase linearly with m, which validates our complexity analysis in § 4.6. In absolute terms, the median computational overhead is less than 12 ms for $m = 1$ and stays below 50 ms for $m = 20$. Note that a complete protocol execution requires identical operations on both sides. To get the total PSI overhead, we can double these numbers if assuming identical hardware for sender and receiver. Still, the PSI operations alone make up less than half of the total authentication delay (cf. Fig. 6). The other major component is networking delay, which we explore next.

6.4 Networking Delay

AirDrop originally uses a wireless connection between sender and receiver. We want to understand the impact of the networking delay and provide a comparison between AWDL and the cable connection (cf. § 6.3). To this end, we repeat the previous experiment over AWDL and measure the transmission delay of the HTTPS requests and replies. In particular, we record timestamps $T_{1..4}$ for each request-response pair, i.e.,

$$T_1 \xrightarrow{\text{Request}} T_2$$
$$T_4 \xleftarrow{\text{Response}} T_3$$

and calculate the delay as $t = T_4 - T_1 - (T_3 - T_2)$ to exclude the receiver-side processing delay. Fig. 8 shows the median transmission delays t incurred by *StartPSI* and *FinishPSI* exchanges for both wireless and cable connections. We add the median transmission delay of AirDrop's *Discover* request for reference. Qualitatively, we can observe that the number of address book entries n has a stronger impact on transmission

delay for AWDL than for the cable connection and that the transmission delay constitutes about *half of the overall authentication delay*. Interestingly, the transmission delay for both PSI requests is similar over the cable, while the first request takes up significantly more time over AWDL. The reason is that the first request includes the time required for connection setup, which generally takes longer over AWDL and has a higher variance, as we discuss next.

High Variance of AWDL Transmission Delays. We noticed a high variance of the transmission delays over AWDL compared to the cable connection (cf. App. A). This effect can be explained by AWDL's channel allocation mechanism. AWDL initially allocates few time slots for transmissions, i.e., little bandwidth is available, and then dynamically allocates more if there is load on the Wi-Fi interface [90]. Thus, initial Wi-Fi transmissions are deferred to the next available time slot, resulting in uncontrollable delays in the order of one second, which is the length of an AWDL period. The increase of available bandwidth over time also explains why the median transmission delay of the first message (*StartPSI*) is significantly larger than the second one (*FinishPSI*).

6.5 Precomputation

While online performance is most crucial for user experience, the precomputation of the encrypted address book entries u_j must also be manageable on mobile devices. Therefore, we evaluate the runtime of calculating the values u_j during the precomputation phase (cf. Fig. 4). As the runtime linearly depends on n (cf. § 4.6), we run a linear regression on the results from an iPhone 12 to approximate the runtime as $n \times 0.331\,\text{ms}$. We provide the raw results in App. B. We see that even for large address books ($n = 10\,\text{k}$), the single-threaded precomputation takes only 3.31 s. To save battery, mobile devices could defer the precomputations to times when they are charging, e.g., overnight.

7 Related Work

We survey closely related works for private mutual authentication, complete our overview of available PSI protocols in addition to our selection process described in § 4.3, review further secure computation techniques, and discuss other privacy leaks in Apple's wireless ecosystem.

7.1 Private Mutual Authentication

The most closely related work to ours is [96]. The authors devise a mutual authentication protocol similar to [3, 4, 54], but geared towards various discovery services, including the contacts-only mode of Apple AirDrop. Utilizing identity-based encryption (IBE) [19], the AirDrop sender distributes encryptions of its identity under a certain "authorization policy". This policy states that only the contacts of this party can decrypt the identity. The authors also implement and benchmark their approach. On a Nexus 5X smartphone, the private authentication takes 360.4 ms.

First of all, the work of [96] mainly targets a different privacy issue in AirDrop, namely the information leakage caused by exchanging the certificates for establishing the TLS connection, which leaks information even to nearby passive adversaries. However, the authors operate under the assumption that these certificates contain the device owner's identity in the clear and are actually used for verifying that sender and receiver are mutual contacts. As recently shown in [92], this is not how AirDrop is currently implemented: the certificates contain only an account-specific UUID while the contact check takes place after the TLS connection is established by exchanging hash values of contact identifiers.

Another conceptual disadvantage of [96] is that Apple, as the IBE root, must provision secret keys to all AirDrop devices, whereas we only require Apple to sign encryptions of hashed contact identifiers where the key can be chosen by the client. Moreover, the system proposed in [96] does not consider subtle issues related to everyday use cases, e.g., how to handle transfers of phone numbers. This would require additional effort to extend the employed IBE scheme with efficient revocation capabilities [18].

In terms of implementation and evaluation, we provide an actual integration into the AirDrop protocol with prototypes on various state-of-the-art Apple devices and demonstrate practical performance under real-world conditions.

7.2 Private Set Intersection

The study of PSI protocols is a very active field of research with various optimizations for different use cases. The "standard" scenario is two-party PSI with balanced input sets and security against semi-honest adversaries, who honestly follow the protocol but try to learn additional information from the transcript. Here, works based on oblivious transfer [60, 76, 79, 80] define the state-of-the-art in terms of concrete performance, while others consider the cost-efficiency in cloud deployment as the most relevant metric [75]. There have been attempts to translate these works to the malicious model [83, 84] with a recent efficiency break-through [74].

PSI was also studied in the multi-party case [43, 50, 61] and extended to generic protocols that can compute an arbitrary symmetric function on top of the intersection [29, 77, 78].

As discussed in § 4.3, most closely related to the problem studied in our work are so-called unbalanced PSI protocols that work particularly well when one of the input sets is much larger than the other [26, 27, 55, 59, 82]. Chen et al. [26, 27] present protocols based on fully homomorphic encryption that are very computation intensive and thus not suitable to be run between two mobile devices. Kiss et al. [59] and Kales et al. [55] optimize protocols based on oblivious pseudorandom function evaluations for the mobile use case, especially so-called mobile contact discovery to privately synchronize address books with user databases in messaging applications. However, these protocols, in the best case, only consider security against malicious PSI receivers but not senders.

There also exist approaches that efficiently outsource PSI computations to a third-party server [1, 2, 56, 57, 99]. However, such protocols are not suitable for our use case since the input parties might be both offline.

Finally, we observe that purely public key-based Diffie-Hellman-style protocols [13, 31, 33, 52, 53, 82], as have been around since the 80's [67, 87], are viable alternatives given the requirements and specified input sizes. Specifically, [31] and [53] are suitable candidates as they are secure against malicious adversaries. We base our work on [53] as it requires fewer exponentiations than the RSA-based protocol of [31] and can be instantiated more efficiently with elliptic curve cryptography. As described in § 4.5, we augment this protocol with signed inputs to prevent certain attacks on the ideal functionality of PSI, similar to the notion of authorized PSI (APSI) [31, 33] and PSI with certified sets [21].

7.3 Secure Computation Protocols

There exist further generic and specialized cryptographic protocols to securely perform the operations necessary for mutual authentication. We efficiently achieve this via PSI in two rounds with $O(m+n)$ complexity (cf. § 4).

Secure two-party computation protocols proposed by Yao [97] and Goldreich, Micali, and Wigderson [40] can obliviously evaluate arbitrary Boolean or arithmetic circuits over private inputs. However, a naive circuit for performing equality tests on m contact identifiers and n address book entries has complexity $O(m \cdot n)$. This complexity can be reduced to be linear with hashing techniques known from so-called circuit-based PSI [29, 76, 77, 78]. Unfortunately, such hashing techniques are incompatible with malicious security [74], which otherwise can be guaranteed with generic approaches [62, 72, 95] at the cost of additional overhead. Furthermore, it is unclear how to efficiently authenticate the contact identifiers used as inputs. There also exist specialized protocols for securely performing comparisons/equality checks (e.g., [30, 64, 98]).

The task of computing the intersection between two sets can be equivalently formulated as the receiver querying/searching the sender's database on its inputs to test for set membership. This can be done while hiding the query and without transferring the entire database via private information retrieval (PIR). While there exists efficient multi-server PIR [28, 36], we consider a two-party setting and hence a single server. State-of-the-art single-server PIR is based on homomorphic encryption [39, 58, 63], which is computationally too demanding for mobile devices. Moreover, PIR does not necessarily protect unrelated database entries, which in our case should remain private. This setting is addressed by works that allow (complex) search queries on encrypted data [38]. Unfortunately, such systems inherently suffer from a certain leakage and have been prone to attacks [17, 23, 70].

7.4 Privacy of Apple's Wireless Ecosystem

AirDrop is part of Apple's larger wireless ecosystem, which has been analyzed for privacy leaks before. AWDL was found to leak personally identifiable information such as the user's real name [92]. Several works [14, 24, 65] have analyzed Apple's Bluetooth implementation and found various ways of tracking devices via static identifiers in Bluetooth advertisements. Finally, [88] discovered that Apple devices can be tracked via identifiers that are randomized asynchronously.

8 Conclusion

In this paper, we solved the problem of privacy-preserving authentication between offline peers, based on the notion of being mutual contacts. We demonstrated the practicability of our approach via a comprehensive experimental performance evaluation, which attests negligible overhead under real-world conditions. We motivated our work with *two* distinct design flaws in AirDrop that allow attackers to learn the phone numbers and email addresses of both sender and receiver devices. However, our proposed protocol can support other applications, even outside of Apple's ecosystem. For example, Google recently launched a similar platform called "Nearby" for Android [41, 86], where device visibility can be restricted to the user's contacts and thus would benefit from our protocol for privacy-preserving authentication.

Our proposed solution PrivateDrop prevents users from disclosing personal information to non-contacts. Still, users remain trackable via their account-specific UUID in the TLS certificate, which gives room for future work. Nevertheless, our results demonstrate that PSI with malicious security is ready for practical deployment, even in offline scenarios between resource-constrained mobile devices. We would be glad to see our open-source implementation being adopted in end-user systems such as AirDrop.

Responsible Disclosure

We informed the Apple Product Security team about our findings (follow-up ID 705937802): We disclosed the sender identifier leakage (cf. § 3.3) in May 2019 and the receiver identifier leakage (cf. § 3.4) as well as our proposed PSI-based protocol (cf. § 4) in October 2020. Apple has not yet commented if they plan to address these AirDrop issues.

Availability

We open-source our PrivateDrop implementation [45] and the code to reproduce our figures [44] as part of the Open Wireless Link project [91].

Acknowledgments

We thank the anonymous reviewers and our shepherd Wouter Lueks for their valuable comments, Benny Pinkas and Gowri R Chandran for insightful discussions, Oliver Schick for help with Relic, and Nanako Honda for explorative work.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, by the LOEWE initiative (Hesse, Germany) within the emergenCITY center, by the German Federal Ministry of Education and Research and the Hessian State Ministry for Higher Education, Research and the Arts within ATHENE.

References

[1] Aydin Abadi, Sotirios Terzis, and Changyu Dong. "VD-PSI: Verifiable Delegated Private Set Intersection on Outsourced Private Datasets". In: *FC*. Springer, 2016, pp. 149–168.

[2] Aydin Abadi, Sotirios Terzis, Roberto Metere, and Changyu Dong. "Efficient Delegated Private Set Intersection on Outsourced Private Datasets". In: *TDSC* 16.4 (2019), pp. 608–624.

[3] Martín Abadi. "Private Authentication". In: *Privacy Enhancing Technologies*. Springer, 2002, pp. 27–40.

[4] Martín Abadi and Cédric Fournet. "Private Authentication". In: *Theor. Comput. Sci.* 322.3 (2004), pp. 427–476.

[5] Apple Inc. *Apple Reports Record First Quarter Results*. Jan. 28, 2020. URL: https://www.apple.com/newsroom/2020/01/apple-reports-record-first-quarter-results/ (visited on 10/15/2020).

[6] Apple Inc. *Cryptographic Message Syntax Services*. 2020. URL: https://developer.apple.com/documentation/security/cryptographic_message_syntax_services (visited on 10/15/2020).

[7] Apple Inc. *CryptoKit*. 2020. URL: https://developer.apple.com/documentation/cryptokit (visited on 10/15/2020).

[8] Apple Inc. *NetService*. 2020. URL: https://developer.apple.com/documentation/foundation/netservice (visited on 10/15/2020).

[9] Apple Inc. *SwiftNIO*. 2020. URL: https://github.com/apple/swift-nio (visited on 10/15/2020).

[10] Apple Inc. *Use AirDrop on your iPhone, iPad, or iPod touch*. Oct. 2019. URL: https://support.apple.com/en-us/HT204144 (visited on 10/15/2020).

[11] Diego F. Aranha, Conrado P. L. Gouvêa, Tobias Markmann, Riad S. Wahby, and Kevin Liao. *RELIC is an Efficient LIbrary for Cryptography*. URL: https://github.com/relic-toolkit/relic (visited on 10/15/2020).

[12] N. Asokan, Alexandra Dmitrienko, Marcin Nagy, Elena Reshetova, Ahmad-Reza Sadeghi, Thomas Schneider, and Stanislaus Stelle. "CrowdShare: Secure Mobile Resource Sharing". In: *ACNS*. Springer, 2013, pp. 432–440.

[13] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. "Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes". In: *CCS*. ACM, 2011, pp. 691–702.

[14] Johannes K. Becker, David Li, and David Starobinski. "Tracking Anonymized Bluetooth Devices". In: *PoPETs* 2019.3 (2019), pp. 50–65.

[15] Ryad Benadjila, Arnaud Ebalard, and Jean-Pierre Flori. *libecc Project*. URL: https://github.com/ANSSI-FR/libecc (visited on 10/15/2020).

[16] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". In: *ASIACRYPT*. Springer, 2012, pp. 626–643.

[17] Laura Blackstone, Seny Kamara, and Tarik Moataz. "Revisiting Leakage Abuse Attacks". In: *NDSS*. Internet Society, 2020.

[18] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. "Identity-based Encryption with Efficient Revocation". In: *CCS*. ACM, 2008, pp. 417–426.

[19] Dan Boneh and Matthew K. Franklin. "Identity-Based Encryption from the Weil Pairing". In: *CRYPTO*. Springer, 2001, pp. 213–229.

[20] Elie Bursztein, Mike Hamburg, Jocelyn Lagarenne, and Dan Boneh. "OpenConflict: Preventing Real Time Map Hacks in Online Games". In: *S&P*. IEEE, 2011, pp. 506–520.

[21] Jan Camenisch and Gregory M. Zaverucha. "Private Intersection of Certified Sets". In: *FC*. Springer, 2009, pp. 108–127.

[22] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. "The Information Visualizer, an Information Workspace". In: *CHI*. ACM, 1991, pp. 181–186.

[23] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. "Leakage-Abuse Attacks Against Searchable Encryption". In: *CCS*. ACM, 2015, pp. 668–679.

[24] Guillaume Celosia and Mathieu Cunche. "Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols". In: *PoPETs* 2020.1 (2020), pp. 26–46.

[25] Dmitry Chastuhin. *Apple Bleee: Everyone Knows What Happens on Your iPhone*. July 25, 2019. URL: https://hexway.io/research/apple-bleee/ (visited on 10/15/2020).

[26] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. "Labeled PSI from Fully Homomorphic Encryption with Malicious Security". In: *CCS*. ACM, 2018, pp. 1223–1237.

[27] Hao Chen, Kim Laine, and Peter Rindal. "Fast Private Set Intersection from Homomorphic Encryption". In: *CCS*. ACM, 2017, pp. 1243–1255.

[28] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. "Private Information Retrieval". In: *FOCS*. IEEE, 1995, pp. 41–50.

[29] Michele Ciampi and Claudio Orlandi. "Combining Private Set-Intersection with Secure Two-Party Computation". In: *SCN*. Springer, 2018, pp. 464–482.

[30] Geoffroy Couteau. "New Protocols for Secure Equality Test and Comparison". In: *ACNS*. Springer, 2018, pp. 303–320.

[31] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. "Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model". In: *ASIACRYPT*. Springer, 2010, pp. 213–231.

[32] Emiliano De Cristofaro, Mark Manulis, and Bertram Poettering. "Private Discovery of Common Social Contacts". In: *ACNS*. Springer, 2011, pp. 147–165.

[33] Emiliano De Cristofaro and Gene Tsudik. "Practical Private Set Intersection Protocols with Linear Complexity". In: *FC*. Springer, 2010, pp. 143–159.

[34] Datafinder. *Recover Encrypted Email Addresses*. 2020. URL: https://web.archive.org/web/20191211152224/https://datafinder.com/products/email-recovery (visited on 10/15/2020).

[35] Levent Demir, Amrit Kumar, Mathieu Cunche, and Cédric Lauradoux. "The Pitfalls of Hashing for Privacy". In: *Commun. Surv. Tutorials* 20.1 (2018), pp. 551–565.

[36] Daniel Demmler, Amir Herzberg, and Thomas Schneider. "RAID-PIR: Practical Multi-Server PIR". In: *CCSW*. ACM, 2014, pp. 45–56.

[37] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO*. Springer, 1986, pp. 186–194.

[38] Benjamin Fuller, Mayank Varia, Arkady Yerand Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. "SoK: Cryptographically Protected Database Search". In: *S&P*. IEEE, 2017, pp. 172–191.

[39] Craig Gentry and Shai Halevi. "Compressible FHE with Applications to PIR". In: *TCC*. Springer, 2019, pp. 438–464.

[40] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *STOC*. ACM, 1987, pp. 218–229.

[41] Google Developers. *Nearby - A platform for discovering and communicating with nearby devices*. URL: https://developers.google.com/nearby (visited on 10/15/2020).

[42] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. "All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers". In: *NDSS*. Internet Society, 2021.

[43] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. "Scalable Multi-party Private Set-Intersection". In: *PKC*. Springer, 2017, pp. 175–203.

[44] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. *PrivateDrop Evaluation*. URL: https://github.com/seemoo-lab/privatedrop-evaluation.

[45] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. *PrivateDrop Implementation*. URL: https://github.com/seemoo-lab/privatedrop.

[46] Alexander Heinrich and Milan Stute. *OpenDrop: an Open Source AirDrop Implementation*. URL: https://github.com/seemoo-lab/opendrop (visited on 10/15/2020).

[47] Russell Housley. "Cryptographic Message Syntax (CMS)". In: *RFC* 5652 (Sept. 2009). DOI: 10.17487/RFC5652.

[48] Troy Hunt. *Have I Been Pwned*. URL: https://haveibeenpwned.com (visited on 10/15/2020).

[49] Kent Ickler. *Hashcat Benchmarks for Nvidia GTX 1080TI*. June 20, 2017. URL: https://www.blackhillsinfosec.com/hashcat-benchmarks-nvidia-gtx-1080ti-gtx-1070-hashcat-benchmarks/ (visited on 10/15/2020).

[50] Roi Inbar, Eran Omri, and Benny Pinkas. "Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters". In: *SCN*. Springer, 2018, pp. 235–252.

[51] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. "On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality Protocols". In: *EuroS&P*. IEEE, 2020.

[52] Stanislaw Jarecki and Xiaomin Liu. "Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection". In: *TCC*. Springer, 2009, pp. 577–594.

[53] Stanislaw Jarecki and Xiaomin Liu. "Fast Secure Computation of Set Intersection". In: *SCN*. Springer, 2010, pp. 418–435.

[54] Stanislaw Jarecki and Xiaomin Liu. "Private Mutual Authentication and Conditional Oblivious Transfer". In: *CRYPTO*. Springer, 2009, pp. 90–107.

[55] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. "Mobile Private Contact Discovery at Scale". In: *USENIX Security*. USENIX Association, 2019, pp. 1447–1464.

[56] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. "Scaling Private Set Intersection to Billion-Element Sets". In: *FC*. Springer, 2014, pp. 195–215.

[57] Florian Kerschbaum. "Outsourced Private Set Intersection Using Homomorphic Encryption". In: *AsiaCCS*. ACM, 2012, pp. 85–86.

[58] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. "Optimal Rate Private Information Retrieval from Homomorphic Encryption". In: *PoPETs* 2015.2 (2015), pp. 222–243.

[59] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. "Private Set Intersection for Unequal Set Sizes with Mobile Applications". In: *PoPETs* 2017.4 (2017), pp. 177–197.

[60] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *CCS*. ACM, 2016, pp. 818–829.

[61] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. "Practical Multi-party Private Set Intersection from Symmetric-Key Techniques". In: *CCS*. ACM, 2017, pp. 1257–1272.

[62] Yehuda Lindell. "Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries". In: *CRYPTO*. Springer, 2013, pp. 1–17.

[63] Helger Lipmaa and Kateryna Pavlyk. "A Simpler Rate-Optimal CPIR Protocol". In: *FC*. Springer, 2017, pp. 621–638.

[64] Helger Lipmaa and Tomas Toft. "Secure Equality and Greater-Than Tests with Sublinear Online Complexity". In: *ICALP*. Springer, 2013, pp. 645–656.

[65] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik C. Rye, Brandon Sipes, and Sam Teplov. "Handoff All Your Privacy - A Review of Apple's Bluetooth Low Energy Continuity Protocol". In: *PoPETs* 2019.4 (2019), pp. 34–53.

[66] Matthias Marx, Ephraim Zimmer, Tobias Mueller, Maximilian Blochberger, and Hannes Federrath. "Hashing of Personally Identifiable Information is not Sufficient". In: *Sicherheit*. Vol. P-281. LNI. GI e.V., 2018, pp. 55–68.

[67] Catherine A. Meadows. "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party". In: *S&P*. IEEE, 1986, pp. 134–137.

[68] MIRACL UK Ltd. *MIRACL – Multiprecision Integer and Rational Arithmetic Cryptographic Library*. URL: https://github.com/miracl/MIRACL (visited on 10/15/2020).

[69] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Tech. rep. Aug. 2015.

[70] Muhammad Naveed, Seny Kamara, and Charles V. Wright. "Inference Attacks on Property-Preserving Encrypted Databases". In: *CCS*. ACM, 2015, pp. 644–655.

[71] OpenSSL Software Foundation. *OpenSSL: Cryptography and SSL/TLS Toolkit*. URL: https://www.openssl.org (visited on 10/15/2020).

[72] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. "Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption". In: *CT-RSA*. Springer, 2020, pp. 254–283.

[73] Daniel Pigatto, Natassya Silva, and Kalinka Castelo Branco. "Performance Evaluation and Comparison of Algorithms for Elliptic Curve Cryptography with El-Gamal based on MIRACL and RELIC Libraries". In: *Journal of Applied Computing Research* 112 (2011).

[74] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. "PSI from PaXoS: Fast, Malicious Private Set Intersection". In: *EUROCRYPT*. Springer, 2020, pp. 739–767.

[75] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. "SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension". In: *CRYPTO*. Springer, 2019, pp. 401–431.

[76] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. "Phasing: Private Set Intersection Using Permutation-based Hashing". In: *USENIX Security*. USENIX Association, 2015, pp. 515–530.

[77] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. "Efficient Circuit-Based PSI with Linear Communication". In: *EUROCRYPT*. Springer, 2019, pp. 122–153.

[78] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. "Efficient Circuit-Based PSI via Cuckoo Hashing". In: *EUROCRYPT*. Springer, 2018, pp. 125–157.

[79] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Faster Private Set Intersection Based on OT Extension". In: *USENIX Security*. USENIX Association, 2014, pp. 797–812.

[80] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Scalable Private Set Intersection Based on OT Extension". In: *TOPS* 21.2 (2018), 7:1–7:35.

[81] Lucian Popa, Bogdan Groza, and Pal-Stefan Murvay. "Performance Evaluation of Elliptic Curve Libraries on Automotive-Grade Microcontrollers". In: *ARES*. ACM, 2019, 100:1–100:7.

[82] Amanda C. Davi Resende and Diego F. Aranha. "Faster Unbalanced Private Set Intersection". In: *FC*. Springer, 2018, pp. 203–221.

[83] Peter Rindal and Mike Rosulek. "Improved Private Set Intersection Against Malicious Adversaries". In: *EUROCRYPT*. Springer, 2017, pp. 235–259.

[84] Peter Rindal and Mike Rosulek. "Malicious-Secure Private Set Intersection via Dual Execution". In: *CCS*. ACM, 2017, pp. 1229–1242.

[85] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO*. Springer, 1989, pp. 239–252.

[86] Daniel Marcos Schwaycer. *Instantly share files with people around you with Nearby Share*. Aug. 2020. URL: https://blog.google/products/android/nearby-share/ (visited on 10/15/2020).

[87] Adi Shamir. "On the Power of Commutativity in Cryptography". In: *ICALP*. Springer, 1980, pp. 582–595.

[88] Milan Stute, Alexander Heinrich, Jannik Lorenz, and Matthias Hollick. "Disrupting Continuity of Apple's Wireless Ecosystem Security: New Tracking, DoS, and MitM Attacks on iOS and macOS Through Bluetooth Low Energy, AWDL, and Wi-Fi". In: *USENIX Security*. To appear. USENIX Association, 2021.

[89] Milan Stute, David Kreitschmann, and Matthias Hollick. "Linux Goes Apple Picking: Cross-Platform Ad hoc Communication with Apple Wireless Direct Link". In: *MobiCom*. ACM, 2018, pp. 820–822.

[90] Milan Stute, David Kreitschmann, and Matthias Hollick. "One Billion Apples' Secret Sauce: Recipe for the *Apple Wireless Direct Link* Ad hoc Protocol". In: *MobiCom*. ACM, 2018, pp. 529–543.

[91] Milan Stute, David Kreitschmann, and Matthias Hollick. *The Open Wireless Link Project*. 2018. URL: https://owlink.org.

[92] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. "A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link". In: *USENIX Security*. USENIX Association, 2019, pp. 37–54.

[93] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. "Protecting accounts from credential stuffing with password breach alerting". In: *USENIX Security*. USENIX Association, 2019, pp. 1556–1571.

[94] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. "Epione: Lightweight Contact Tracing with Strong Privacy". In: *Data Eng. Bull.* 43.2 (2020), pp. 95–107.

[95] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. "Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation". In: *CCS*. ACM, 2017, pp. 21–37.

[96] David J. Wu, Ankur Taly, Asim Shankar, and Dan Boneh. "Privacy, Discovery, and Authentication for the Internet of Things". In: *ESORICS*. Springer, 2016, pp. 301–319.

[97] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets". In: *FOCS*. IEEE, 1986, pp. 162–167.

[98] Ching-Hua Yu and Bo-Yin Yang. "Probabilistically Correct Secure Arithmetic Computation for Modular Conversion, Zero Test, Comparison, MOD and Exponentiation". In: *SCN*. Springer, 2012, pp. 426–444.

[99] Qingji Zheng and Shouhuai Xu. "Verifiable Delegated Set Intersection Operations on Outsourced Encrypted Data". In: *IC2E*. IEEE, 2015, pp. 175–184.

A Authentication Delay over AWDL

Fig. 9 shows the high variance of the authentication delay of PrivateDrop over the AWDL interface. The lower and upper error bars indicate the 0.05 and 0.95 quantiles, respectively. Still, the median authentication delay for PrivateDrop lies within 500 ms and 1 500 ms, depending on (m,n).

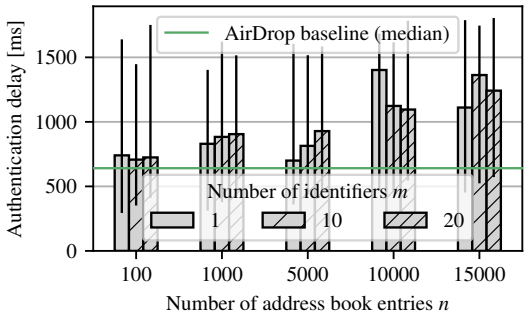


Figure 9: Overall authentication delay for AirDrop (baseline) and PrivateDrop with different set sizes (m,n) (MacBook Pro 2019 → iPhone 12 via AWDL).

B PSI Precomputation

Fig. 10 shows the runtime of the PSI precomputation required for calculating u_i (cf. precomputation phase in Fig. 4) on an iPhone 12. Even with a large address book ($n = 15\,000$), the computation time does not exceed 5 s, which is very manageable for a mobile device that charges overnight.

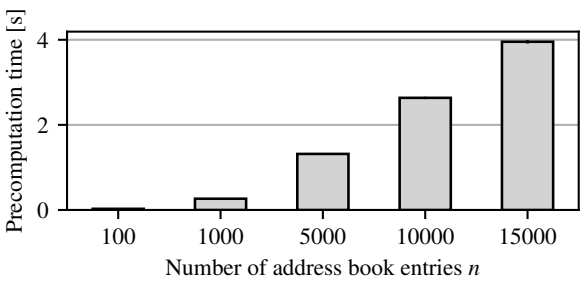


Figure 10: Runtime of PSI precomputation on an iPhone 12.

C Performance Comparison with Apple's AirDrop Implementation

We benchmark our base AirDrop implementation against Apple's original one. To evaluate Apple's implementation, we leverage the system logging facility of macOS (cf. [88]) that produces debug output for AirDrop and provides logs verbose enough to distinguish the authentication phase. We calculate the authentication delay as the timestamp difference of the entries indicating the start and end of the authentication phase. We provide the details in our evaluation repository [44]. We use the same hardware configuration and environment as described in § 6.2. We open the sharing pane on the MacBook Pro and manually wake up the iPhone 12 by tapping on the screen. We repeat this process 100 times and report on the results in Fig. 11 as an empirical cumulative distribution function. The results show that the best-case performance of our implementation is similar to the original one. The high variance of the delay can be attributed to the initialization behavior of AWDL (cf. § 6.4).

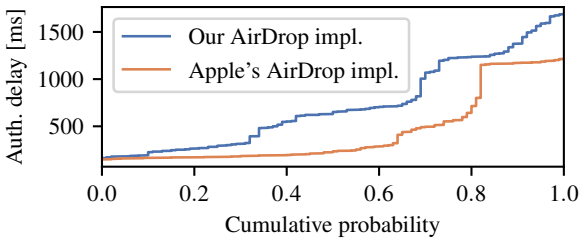


Figure 11: Authentication delay of our AirDrop implementation and Apple's (MacBook Pro 2019 → iPhone 12 via AWDL).

E Efficient Circuit-Based PSI via Cuckoo Hashing (EUROCRYPT'18)

[PSWW18] B. PINKAS, T. SCHNEIDER, C. WEINERT, U. WIEDER. **"Efficient Circuit-Based PSI via Cuckoo Hashing"**. In: *37. Advances in Cryptology – EUROCRYPT'18*. Vol. 10822. LNCS. Code: https://encrypto.de/code/2DCH. Full version: https://ia.cr/2018/120. Springer, 2018, pp. 125–157. CORE Rank A*. Appendix E.

https://doi.org/10.1007/978-3-319-78372-7_5

Efficient Circuit-Based PSI via Cuckoo Hashing

Benny Pinkas[1( )], Thomas Schneider[2], Christian Weinert[2], and Udi Wieder[3]

[1] Bar-Ilan University, Ramat Gan, Israel
benny@pinkas.net
[2] TU Darmstadt, Darmstadt, Germany
{thomas.schneider,christian.weinert}@crisp-da.de
[3] VMware Research, Palo Alto, USA
udi.wieder@gmail.com

Abstract. While there has been a lot of progress in designing efficient custom protocols for computing Private Set Intersection (PSI), there has been less research on using generic Multi-Party Computation (MPC) protocols for this task. However, there are many variants of the set intersection functionality that are not addressed by the existing custom PSI solutions and are easy to compute with generic MPC protocols (e.g., comparing the cardinality of the intersection with a threshold or measuring ad conversion rates).

Generic PSI protocols work over circuits that compute the intersection. For sets of size n, the best known circuit constructions conduct $O(n \log n)$ or $O(n \log n / \log \log n)$ comparisons (Huang et al., NDSS'12 and Pinkas et al., USENIX Security'15). In this work, we propose new circuit-based protocols for computing *variants of the intersection* with an almost linear number of comparisons. Our constructions are based on new variants of Cuckoo hashing in two dimensions.

We present an asymptotically efficient protocol as well as a protocol with better concrete efficiency. For the latter protocol, we determine the required sizes of tables and circuits experimentally, and show that the run-time is concretely better than that of existing constructions.

The protocol can be extended to a larger number of parties. The proof technique presented in the full version for analyzing Cuckoo hashing in two dimensions is new and can be generalized to analyzing standard Cuckoo hashing as well as other new variants of it.

Keywords: Private set intersection · Secure computation

1 Introduction

Private Set Intersection (PSI) refers to a protocol which enables two parties, holding respective input sets X and Y, to compute the intersection $X \cap Y$ without revealing any information about the items which are not in the intersection. The PSI functionality is useful for applications where parties need to apply a JOIN operation to private datasets. There are multiple constructions of secure

J. B. Nielsen and V. Rijmen (Eds.): EUROCRYPT 2018, LNCS 10822, pp. 125–157, 2018.
https://doi.org/10.1007/978-3-319-78372-7_5

protocols for computing PSI, but there is an advantage for computing PSI by applying a generic Multi-Party Computation (MPC) protocol to a circuit computing the intersection (see Sect. 1.1). The problem is that a naive circuit computes $O(n^2)$ comparisons, and even the most recent circuit-based constructions require $O(n \log n)$ or $O(n \log n / \log \log n)$ comparisons (see Sect. 1.4).

In this work, we present a new circuit-based protocol for computing PSI variants. In our protocol, each party first inserts its input elements into bins according to a new hashing algorithm, and then the intersection is computed by securely computing a Boolean comparison circuit over the bins. The insertion of the items is based on new Cuckoo hashing variants which guarantee that if the two parties have the same input value, then there is exactly one bin to which both parties map this value. Furthermore, the total number of bins is $O(n)$ and there are $O(1)$ items mapped to each bin, plus $\omega(1)$ items which are mapped to a special stash. Hence, the circuit that compares (1) for each bin, the items that the two parties mapped to it, and (2) all stash items to all items of the other party, computes only $\omega(n)$ comparisons.

1.1 Motivation for Circuit-Based PSI

PSI has many applications, as is detailed for example in [42]. Consequently, there has been a lot of research on efficient secure computation of PSI, as we describe in Sect. 1.4. However, most research was focused on computing the intersection itself, while there are interesting applications for the ability to securely compute arbitrary functions of the intersection. We demonstrate the need for efficient computation of PSI using generic protocols through the following arguments:

Adaptability. Assume that you are a cryptographer and were asked to propose and implement a protocol for computing PSI. One approach is to use a specialized protocol for computing PSI. Another possible approach is to use a protocol for generic secure computation, and apply it to a circuit that computes PSI. A trivial circuit performs $O(n^2)$ comparisons, while more efficient circuits, described in [26, 39], perform only $O(n \log n)$ or $O(n \log n / \log \log n)$ comparisons, respectively. The most efficient specialized PSI protocols are faster by about two orders of magnitude than circuit-based constructions (see [39]), and therefore you will probably choose to use a specialized PSI protocol. However, what happens if you are later asked to change the protocol to compute another function of the intersection? For example, output only the size of the intersection, or output 1 iff the size is greater than some threshold, or output the most "representative" item that occurs in the intersection (according to some metric). Any change to a specialized protocol will require considerable cryptographic know-how, and might not even be possible. On the other hand, the task of writing a new circuit component that computes a different function of the intersection is rather trivial, and can even be performed by undergrad students.

Consider the following function as an example of a variant of the PSI functionality for which we do not know a specialized protocol: Suppose that you want to compute the size of the intersection, but you also wish to preserve the privacy

of users by ensuring differential privacy. This is done by adding some noise to the exact count before releasing it. This functionality can easily be computed by a circuit, but it is unclear how to compute it using other PSI protocols. (See [38] for constructions that add noise to the results of MPC computation in order to ensure differential privacy.)

Existing code base. Circuit-based protocols benefit from all the work that was invested in recent years in designing, implementing, and optimizing very efficient systems for generic secure computation. Users can download existing secure computation software, e.g., [13,27], and only need to design the circuit to be computed and implement the appropriate hashing technique.

Existing applications. There are existing applications that need to compute functions over the results of the set intersection. For example, Google reported [34,49] a PSI-based application for measuring ad conversion rates, namely the revenues from ad viewers who later perform a related transaction. This computation can be done by comparing the list of people who have seen an ad with those who have completed a transaction. These lists are held by the advertiser (say, Google or Facebook), and by merchants, respectively. A simple (non-private) solution is for one side to disclose its list of customers to the other side, which computes the necessary statistics. Another option is to run a secure computation over the results of the set intersection. For example, the merchant inputs pairs of the customer-identity and the value of the transactions made by this customer, and the computation calculates the total revenue from customers who have seen an ad, namely customers in the intersection of the sets known to the advertiser and the merchant. Google reported implementing this computation using a Diffie-Hellman-based PSI cardinality protocol (for computing the cardinality of the intersection) and Paillier encryption (for computing the total revenues) [28]. This protocol reveals the identities of the items in the intersection, and seems less efficient than our protocol as it uses public key operations, rather than efficient symmetric cryptographic operations.[1]

1.2 Our Contributions

This work provides the following contributions:

Circuit-based PSI protocols with almost linear overhead. We show a new circuit-based construction for computing any symmetric function on top of PSI, with an asymptotic overhead of only $\omega(n)$ comparisons. (More accurately, for any function $f \in \omega(n)$, the overhead of the construction is $o(f(n))$.) This construction is based on standard Cuckoo hashing.

[1] Facebook is running a computation of this type with companies that have transaction records for a large part of loyalty card holders in the US. According to the report in https://www.eff.org/deeplinks/2012/09/deep-dive-facebook-and-datalogix-whats-actually-getting-shared-and-how-you-can-opt, the computation is done using an insecure PSI variant based on creating pseudonyms using naive hashing of the items.

Small constants. Standard measures of asymptotic security are not always a good reflection of the actual performance on reasonable parameters. Therefore, in addition to the asymptotic improvement, we also show a concrete circuit-based PSI construction. This construction is based on a new variant of Cuckoo hashing, *two-dimensional Cuckoo hashing*, that we introduce in this work. We carefully handle implementation issues to improve the actual overhead of our protocols, and make sure that all constants are small. In particular, we ran extensive experiments to analyze the failure probabilities of the hashing scheme, and find the exact parameters that reduce this statistical failure probability to an acceptable level (e.g., 2^{-40}). Our analysis of the concrete complexities is backed by extensive experiments, which consumed about 5.5 million core hours on the Lichtenberg high performance computer of the TU Darmstadt and were used to set the parameters of the hashing scheme. Given these parameters we implemented the circuit-based PSI protocol and tested it.

Implementation and experiments. We implemented our protocols using the ABY framework for secure two-party computation [13]. Our experiments show that our protocols are considerably faster than the previously best circuit-based constructions. For example, for input sets of $n = 2^{20}$ elements of arbitrary bitlength, we improve the circuit size over the best previous construction by up to a factor of 3.8x.

New Cuckoo hashing analysis. Our two-dimensional Cuckoo hashing is based on a new Cuckoo hashing scheme that employs two tables and each item is mapped to either *two* locations in the first table, or *two* locations in the second table. This is a new Cuckoo hashing variant that has not been analyzed before. In addition to measuring its performance using simulations, we provide a probabilistic analysis of its performance. Interestingly, this analysis can also be used as a new proof technique for the success probability of standard Cuckoo hashing.

1.3 Computing Symmetric Functions

A trivial circuit for PSI that performs $O(n^2)$ comparisons between all pairs of the input items of the two parties allows the parties to set their inputs in any arbitrary order. On the other hand, there exist more efficient circuit-based PSI constructions where each party first independently orders its inputs according to some predefined algorithm: the sorting network-based construction of [26] requires each party to sort its input to the circuit, while the hashing-based construction of [39] requires the parties to map their inputs to bins using some public hash functions. (These constructions are described in Sect. 1.4.) The location of each input item thus depends on the identity of the other inputs of the input owner, and must therefore be kept hidden from the other party.

In this work, we focus on constructing a circuit that computes the intersection. The outputs of this circuit can be the items in the intersection, or some functions of the items in the intersection: say, a "1" for each intersecting item, or an arbitrary function of some data associated with the item (for example, if the items are transactions, we might want to output a financial value associated

with each transaction that appears in the intersection). On top of that circuit it is possible to add circuits for computing any function that is based on the intersection. In order to preserve privacy, the output of that function must be a *symmetric* function of the items in the intersection. Namely, the output of the function must not depend on the *order* of its inputs. There are many examples of interesting symmetric functions of the intersection. (In fact, it is hard to come up with examples for interesting non-symmetric functions of the intersection, except for the intersection itself.) Examples of symmetric functions include:

- Computing the size of the intersection, i.e., PSI cardinality (PSI-CA).
- Computing a threshold function that is based on the size of the intersection. For example, outputting "1" if the size of the intersection is greater than some threshold (PSI-CAT), or outputting a rounded value of the percentage of items that are in the intersection. An extension of PSI-CAT, where the intersection is revealed only if the size of the intersection is greater than a threshold, can be used for privacy-preserving ridesharing [23].
- Computing the size of the intersection while preserving the privacy of users by ensuring differential privacy [17]. This can be done by adding some noise to the exact count.
- Computing the sum of values associated with the items in the intersection. This is used for measuring ad-generated revenue (cf. Sect. 1.1). Similarly, there could be settings where each party associates a value with each transaction, and the output is the sum of the differences between these assigned values in the intersection, or the sum of the squares of the differences, etc.

The circuits for computing all these functions are of size $O(n)$. Therefore, with our new construction, the total size of the circuits for computing these functions is $\omega(n)$, whereas circuit-based PSI protocols [26,39] had size $O(n \log n)$.

If one wishes to compute a function that is not symmetric, or wishes to output the intersection itself, then the circuit must first shuffle the values in the intersection (in order to assign a random location to each item in the intersection) and then compute the function over the shuffled values, or output the shuffled intersection. A circuit for this "shuffle" step has size $O(n \log n)$, as described in [26]. (It is unclear, though, why a circuit-based protocol should be used for computing the intersection, since this job can be done much more efficiently by specialized protocols, e.g., [31,42].)

1.4 Related Work

PSI. Work on protocols for private set intersection was presented as early as [35,46], which introduced public key-based protocols using commutative cryptography, namely the Diffie-Hellman function. A survey of PSI protocols appears in [41]. The goal of these protocols is to let one party learn the intersection itself, rather than to enable the secure computation of arbitrary functions of the intersection. Other PSI protocols are based on oblivious polynomial evaluation [20], blind RSA [11], and Bloom filters [16]. Today's most efficient PSI protocols are

based on hashing the items to bins and then evaluating an oblivious pseudo-random function per bin, which is implemented using oblivious transfer (OT) extension. These protocols have linear complexity and were all implemented and evaluated, see, e.g., [31,39,41,42]. In cases where communication cost is a crucial and computation cost is a minor factor, recent solutions based on fully homomorphic encryption represent an interesting alternative [6]. PSI protocols have also been adapted to the special requirements of mobile devices [4,25,30].

Circuit-based PSI. Circuit-based PSI protocols compute the set intersection functionality by running a secure evaluation of a Boolean circuit. These protocols can easily be adapted to compute different variants of the PSI functionality. The straightforward solution to the PSI problem requires $O(n^2)$ comparisons – one comparison for each pair of items belonging to the two parties. Huang et al. [26] designed a circuit for computing PSI based on sorting networks, which computes $O(n \log n)$ comparisons and is of size $O(\sigma n \log n)$, where σ is the bitlength of the inputs. A different circuit, based on the usage of Cuckoo hashing by one party and simple hashing by the other party, was proposed in [39]. The size of that circuit is $O(\sigma n \log n / \log \log n)$. In our work we propose efficient circuits for PSI variants with an asymptotic size of $\omega(\sigma n)$ and better concrete efficiency. We give more details and a comparison of the concrete complexities of circuit-based PSI protocols in Sect. 6.2.

PSI Cardinality (PSI-CA). A specific interesting function of the intersection is its cardinality, namely $|X \cap Y|$, and is referred to as PSI-CA. There are several protocols for computing PSI-CA with linear complexity based on public key cryptography, e.g., [9] which is based on Diffie-Hellman and is essentially a variant of the DH-based PSI protocol of [35,46] (see also references given therein for other less efficient public key-based protocols); or [12] which is based on Bloom filters and the public key cryptosystem of Goldwasser-Micali. In these protocols, one of the parties learns the cardinality. As we show in our experiments in Sect. 6.3, these protocols are slower than our constructions already for relatively small set sizes ($n = 2^{12}$) in the LAN setting and for large set sizes ($n = 2^{20}$) in the WAN setting, since they are based on public key cryptography. An advantage of these protocols is that they achieve the lowest amount of communication, but it seems hard to extend them to compute arbitrary functions of the intersection. Protocols for private set intersection and union and their cardinalities with linear complexity are given in [8]. They use Bloom filters and computationally expensive additively homomorphic encryption, whereas our protocols can flexibly be adapted to different variants and are based on efficient symmetric key cryptography.

2 Preliminaries

Setting. We consider two parties, which we denote as Alice and Bob. They have input sets, X and Y, respectively, which are each of size n and each item

has bitlength σ. We assume that both parties agree on a symmetric function f and would like to securely compute $f(X \cap Y)$. They also agree on a circuit that receives the items in the intersection as input and computes f.

Security Model. The secure computation literature considers *semi-honest* adversaries, which try to learn as much information as possible from a given protocol execution, but are not able to deviate from the protocol steps, and *malicious* adversaries, which are able to deviate arbitrarily from the protocol. The semi-honest adversary model is appropriate for scenarios where execution of the intended software is guaranteed via software attestation or business restrictions, and yet an untrusted third party is able to obtain the transcript of the protocol after its execution, either by stealing it or by legally enforcing its disclosure. Most protocols for private set intersection, as well as this work, focus on solutions that are secure against semi-honest adversaries. PSI protocols for the malicious setting exist, but they are less efficient than protocols for the semi-honest setting (see, e.g., [7,10,19,20,43,44]).

Secure Computation. There are two main approaches for generic secure two-party computation with security against semi-honest adversaries that allow to securely evaluate a function that is represented as a Boolean circuit: (1) Yao's garbled circuit protocol [48] has a constant round complexity and with today's most efficient optimizations provides free XOR gates [33], whereas securely evaluating an AND gate requires sending two ciphertexts [50]. (2) The GMW protocol [21] also provides free XOR gates and requires two ciphertexts of communication per AND gate using OT extension [3]. The main advantage of the GMW protocol is that *all* symmetric cryptographic operations can be pre-computed in a constant number of rounds in a setup phase, whereas the online phase is very efficient, but requires interaction for each layer of AND gates. In more detail, the setup phase is independent of the actual inputs and pre-computes multiplication triples for each AND gate using OT extension in a constant number of rounds (cf. [3]). The online phase runs from the time the inputs are provided until the result is obtained and involves sending one message for each layer of AND gates. A detailed description and a comparison between Yao and GMW is given in [45].

Cuckoo Hashing. In its simplest form, Cuckoo hashing [36] uses two hash functions h_0, h_1 to map n elements to two tables T_0, T_1, each containing $(1 + \varepsilon)n$ bins. Each bin accommodates at most a single element. The scheme avoids collisions by relocating elements when a collision is found using the following procedure: Let $b \in \{0, 1\}$. An element x is inserted into a bin $h_b(x)$ in table T_b. If a prior item y exists in that bin, it is evicted to bin $h_{1-b}(y)$ in T_{1-b}. The pointer b is then assigned the value $1 - b$. The procedure is repeated until no more evictions are necessary, or until a threshold number of relocations has been performed. In the latter case, the last element is mapped to a special stash. It was shown in [29] that, for any constant s, the probability that the size of the

stash is greater than s is at most $O(n^{-(s+1)})$. After inserting all items, each item can be found in one of two locations or in the stash. A lookup therefore requires checking only $O(1)$ locations.

Many variants of Cuckoo hashing were suggested and analyzed. See [47] for a thorough discussion and analysis of different Cuckoo hashing schemes. A variant of Cuckoo hashing that is similar to our constructions was given in [1], although in a different application domain. It considers a setting with three tables, where an item must be placed in two out of three tables. The analysis of this construction uses a different proof technique than the one we present in the full version [40], and we have not attempted to generalize their proof to a general number of item insertions (as we do for our construction). Furthermore, there is no tight analysis of the stash size in [1]. The work in [18] builds on the construction of [1] and proves that the failure probability when using a stash of size s behaves as $\tilde{O}(n^{-s})$. However, the experiments of [18, Fig. 6] reveal that the size of the stash is rather large and actually *increasing* in n within the range of 1 000 to 100 000 elements. For example, for table size $7.1n$, a stash of at least size 4 is required for inserting 10 000 elements, whereas a stash of at least size 11 is required for inserting 100 000 elements. Since each item in the stash must be compared to all items of the other party, and since these comparisons cannot use a shorter representation based on permutation-based hashing, the effect of the stash is substantial, and in the context of circuit-based PSI it is therefore preferable to use constructions that place very few or no items in the stash.

PSI based on Hashing. Some existing constructions of circuits for PSI require the parties to reorder their inputs before inputting them to the circuit: The sorting-network based construction of [26] requires the parties to sort their inputs. The hashing based construction of [39] requires that each party maps its items to bins using a hash function. It was observed as early as [20] that if the two parties agree on the same hash function and use it to map their respective input to bins, then the items that one party maps to a specific bin need to be compared only to the items that the other party maps to the same bin. However, the parties must be careful not to reveal to each other the number of items they mapped to each bin, since this data leaks information about their other items. Therefore, they agree beforehand on an upper bound m for the maximum number of items that can be mapped to a bin (such upper bounds are well known for common hashing algorithms, and can also be substantiated using simulation), and pad each bin with random dummy values until it has exactly m items in it. If both parties use the same hash algorithm, then this approach considerably reduces the overhead of the computation from $O(n^2)$ to $O(\beta \cdot m^2)$, where m is the maximum number of items mapped to any of the β bins.

When a random hash function h is used to map n items to n bins, where x is mapped to bin $h(x)$, the most occupied bin has w.h.p. $m = \frac{\ln n}{\ln \ln n}(1 + o(1))$ items [22] (a careful analysis shows, e.g., that, for $n = 2^{20}$ and an error probability of 2^{-40}, one needs to set $m = 20$). Cuckoo hashing is much more promising, since it maps n items to $2(1+\varepsilon)n$ bins, where each bin stores at most

$m = 1$ items. Cuckoo hashing typically uses two hash functions h_0, h_1, where an item x is mapped to one of the two locations $h_0(x), h_1(x)$, or to a stash of a small size. It is tempting to let both parties, Alice and Bob, map their items to bins using Cuckoo hashing, and then only compare the item that one party maps to a bin with the item that the other party maps to the same bin. The problem is that Alice might map x to $h_0(x)$ whereas Bob might map it to $h_1(x)$. They cannot use a protocol where Alice's value in bin $h_0(x)$ is compared to the two bins $h_0(x), h_1(x)$ in Bob's input, since this reveals that Alice has an item that is mapped to these two locations. The solution used in [19,39,41] is to let Alice map her items to bins using Cuckoo hashing, and Bob map his items using simple hashing. Namely, each item of Bob is mapped to both bins $h_0(x), h_1(x)$. Therefore, Bob needs to pad his bins to have $m = O(\log n / \log \log n)$ items in each bin, and the total number of comparisons is $O(n \log n / \log \log n)$.

3 Analyzing the Failure Probability

Efficient cryptographic protocols that are based on probabilistic constructions are typically secure as long as the underlying probabilistic constructions do not fail. Our work is based on variants of Cuckoo hashing, and the protocols are secure as long as the relevant tables and stashes do not overflow. (Specifically, hashing is computed using random hash functions which are chosen independently of the data. If a party observes that these functions cannot successfully hash its data, it can indeed ask to replace the hash functions, or remove some items from its input. However, the hash functions are then no longer independent of this party's input and might therefore leak some information about the input.)

There are two approaches for arguing about the failure probability of cryptographic protocols:

1. For an **asymptotic analysis**, the failure probability must be negligible in n.
2. For a **concrete analysis**, the failure probability is set to be smaller than some threshold, say $2^{-\lambda}$, where λ is a statistical security parameter.
   In typical experiments, the statistical security parameter is set to $\lambda = 40$. This means that "unfortunate" events that leak information happen with a probability of at most 2^{-40}. In particular, $\lambda = 40$ was used in all PSI constructions which are based on hashing (e.g., [16,19,31,39,41]).

With regards to the probabilistic constructions, there are different levels of analysis of the failure probability:

1. For simple constructions, it is sometimes possible to compute the **exact failure probability**. (For example, suppose that items are hashed to a table using a random hash function, and a failure happens when two items are mapped to the same location. In this case it is trivial to compute the exact failure probability.)

2. For some constructions there are known **asymptotic bounds** for the failure probability, but no concrete expressions. (For example, for Cuckoo hashing with a stash of size s, it was shown in [29] that the overflow probability is $O(n^{-(s+1)})$, but the exact constants are unknown.)[2]
3. For other constructions there is no analysis for the failure probability, even though they **perform very well in practice**. For example, Cuckoo hashing variants where items can be mapped to $d > 2$ locations, or where each bin can hold $k > 1$ items, were known to have better space utilization than standard Cuckoo hashing, but it took several years to theoretically analyze their performance [47]. There are also insertion algorithms for these Cuckoo hashing variants which are known to perform well but which have not yet been fully analyzed.

3.1 Using Probabilistic Constructions for Cryptography

Suppose that one is using a probabilistic construction (e.g., a hash table) in the design of a cryptographic protocol. An asymptotic analysis of the cryptographic protocol can be done if the hash table has either an exact analysis or an asymptotic analysis of its failure probability (items 1 and 2 in the previous list).

If the aim is a concrete analysis of the cryptographic protocol, then exact values for the parameters of the hash construction must be identified. If an exact analysis is known (item 1), then it is easy to plug in the desired failure probability ($2^{-\lambda}$) and compute the values for the different parameters. However, if only an asymptotic analysis or experimental evidence is known (items 2 and 3), then experiments must be run in order to find the parameters that set the failure probability to be smaller than $2^{-\lambda}$.

We stress that a concrete analysis is needed whenever a cryptographic protocol is to be used in practice. In that case, even an asymptotic analysis is insufficient since it does not specify any constants, which are crucial for deriving the exact parameter values.

3.2 Experimental Parameter Analysis

Verifying that the failure probability is smaller than $2^{-\lambda}$ for $\lambda = 40$ requires running many repetitions of the experiments. Furthermore, for large input sizes (large values of n), each single run of the experiment can be rather lengthy. (And one could justifiably argue that the more interesting results are for the larger values of n, since for smaller n we can use less optimal constructions and still get reasonable performance.)

[2] We note though that many probabilistic constructions are analyzed in the algorithms research literature to have a failure probability of $o(1)$, which is fine for many applications, but is typically insufficient for cryptographic applications.

Examining the failure probability for a specific choice of parameters. For a specific choice of parameters, running 2^λ repetitions of an experiment is insufficient to argue about a $2^{-\lambda}$ failure probability, since it might happen that the experiments were very unlucky and resulted in no failure even though the failure probability is somewhat larger than $2^{-\lambda}$. Instead, we can argue about a confidence interval: namely, a confidence interval of $1-\alpha$ (say, 95%, or 99.9%) states that if the failure probability is greater than $2^{-\lambda}$, then we would have *not* seen the results of the experiment, except with a probability that is smaller than α. Therefore, either the experiment was very unlucky, or the failure probability is sufficiently small. For example, an easy to remember confidence level used in statistics is the "rule of three", which states that if an event has not occurred in $3 \cdot s$ experiments, then the 95% confidence interval for its rate of occurrence in the population is $[0, 1/s]$. For our purposes this means that running $3 \cdot 2^\lambda$ experiments with no failure suffices to state that the failure probability is smaller than $2^{-\lambda}$ with 95% confidence. (We will report experiments in Sect. 6.1 which result in a 99.9% confidence interval for the failure probability.)

Examining the failure probability as a function of n. For large values of n (e.g., $n = 2^{20}$), it might be too costly to run sufficiently many (more than 2^{40}) experiments. Suppose that the experiments spend just 10 cycles on each item. This is an extremely small lower bound, which is probably optimistic by orders of magnitude compared to the actual run-time. Then the experiments take at least $10 \cdot 2^{60}$ cycles. This translates to about a million core hours on 3 GHz machines.

In order to be able to argue about the failure probability for large values of n, we can run experiments for progressively increasing values of n and identify how the failure probability behaves as a function of n. If we observe that the failure probability is decreasing, or, better still, identify the dependence on n, we can argue, given experimental results for medium-sized n values, about the failure probabilities for larger values of n.

3.3 Our Constructions

Asymptotic overhead. We present in Sect. 4 a construction of circuit-based PSI that we denote as the "mirror" construction. This construction uses four instances of standard Cuckoo hashing and therefore we know that a stash of size s guarantees a failure probability of $O(n^{-(s+1)})$ [29]. (Actually, the previously known analysis was only stated for $s = O(1)$. We show in the full version [40] that this failure probability also holds for s that is not constant.)

The bound on the failure probability implies that for any constant security parameter λ, a stash of constant size is sufficient to ensure that the failure probability is smaller than $2^{-\lambda}$ for sufficiently large n. In order to achieve a failure probability that is negligible in n, we can set the stash size s to be slightly larger than $O(1)$, e.g., $s = \log\log n$, $s = \log^* n$, or any $s = \omega(1)$. The result is a construction with an overhead of $\omega(n)$. (More accurately, the overhead is as close as desired to being linear: for any $f(n) \in \omega(n)$, the overhead is $o(f(n))$.)

Concrete overhead. In Sect. 5 we present a new variant of Cuckoo hashing that we denote as two-dimensional (or 2D) Cuckoo hashing. We analyze this construction in the full version [40] and show that when no stash is used, then the failure probability (with tables of size $O(n)$) is $O(1/n)$, as in standard Cuckoo hashing.

We only have a sketch of an analysis for the size of the stash of the construction in Sect. 5, but we observed that this construction performed much better than the asymptotic construction. Also, performance was improved with the heuristic of using half as many bins but letting each bin store two items instead of one. (This variant is known to perform much better also in the case of standard Cuckoo hashing, see [47].)

Since we do not have a theoretical analysis of this construction, we ran extensive experiments in order to examine its performance. These experiments follow the analysis paradigm given in Sect. 3.2, and are described in Sect. 6.1. For a specific ratio between the table size and n, we ran 2^{40} experiments for $n = 2^{12}$ and found that the failure probability is at most 2^{-37} with 99.9% confidence. We also ran experiments for increasing values of n, up to $n = 2^{12}$, and found that the failure probability has linear dependence on n^{-3} (an explanation of this behavior appears in the full version [40]). Therefore, we can argue that for $n \geq 2^{13} = 2 \cdot 2^{12}$ the failure probability is at most $2^{-37} \cdot 2^{-3} = 2^{-40}$.

4 An Asymptotic Construction Through Mirror Cuckoo Hashing

We show here a construction for circuit-based PSI that has an $\omega(n)$ asymptotic overhead. The analysis in this section is not intended to be tight, but rather shows the asymptotic behavior of the overhead.

The analysis is based on a construction which we denote as *mirror Cuckoo hashing* (as the placement of the hash functions that are used in one side is a mirror image of the hash functions of the other side). Hashing is computed in a single iteration. The main advantage of this construction is that it is based on four copies of standard Cuckoo hashing. Therefore, we can apply known bounds on the failure probability of Cuckoo hashing. Namely, applying the result of [29] that the failure probability when using a stash of size s is $O(n^{-(s+1)})$. Given this result, a stash of size $\omega(1)$ guarantees that the failure probability is negligible in n (while a constant stash size guarantees that for sufficiently large n the failure probability is smaller than any constant, and in particular smaller than 2^{-40}). We note that while the known results about the size of the stash are only stated for $s = O(1)$, we show in the full version [40] that the $O(n^{-(s+1)})$ bound on the failure probability also applies to a non-constant stash size.

4.1 Mirror Cuckoo Hashing

We describe a hashing scheme that uses two sets of tables. A left set including tables T_{L}, T_{R}, and a right set including tables T'_{L}, T'_{R}. Each table is also denoted

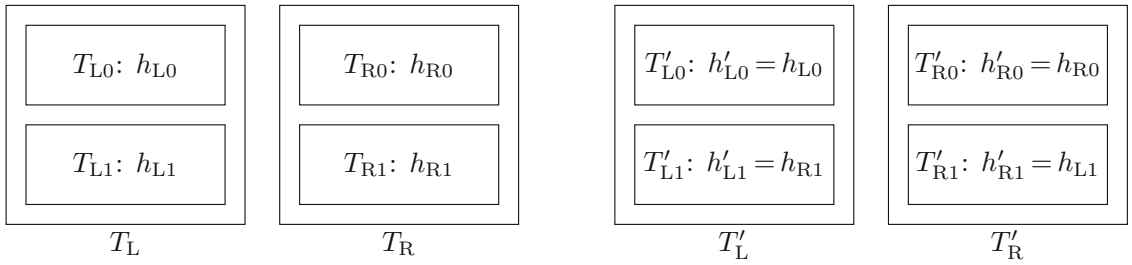


Fig. 1. The tables T_L, T_R and T'_L, T'_R. The hash functions in the upper subtables of T'_L, T'_R are the same as in T_L, T_R, and those in the lower subtables are in reverse order.

as a "column". Each table has two subtables, or "rows". So overall there are four tables (columns), each containing two subtables (rows).

Bob maps each of his items to one subtable in each table, namely to one row in each column. Alice maps each of her items to the two subtables in one of the tables, namely to both rows in just one of the columns. These mappings ensure that for any item x that is owned by both Alice and Bob, there is exactly one subtable to which it is mapped by both parties.

The tables. The construction uses two sets of tables, T_L, T_R and T'_L, T'_R. Each table is of size $2(1+\varepsilon)n$ and is composed of two subtables of size $(1+\varepsilon)n$ (T_L contains the subtables T_{L0}, T_{L1}, etc.). Each subtable is associated with a hash function that will be used by both parties. E.g., function h_{L0} will be used for subtable T_{L0}, etc. The tables and the hash functions are depicted in Fig. 1.

The hash functions. The hash functions associated with the tables are defined as follows:

- The functions for the left two tables (columns) T_L, T_R, i.e., h_{L0}, h_{L1}, h_{R0}, h_{R1}, are chosen at random. Each function maps items to the range $[0, (1+\varepsilon)n-1]$, which corresponds to the number of bins in each of T_{L0}, T_{L1}, T_{R0}, T_{R1}.
- The functions for the two right tables T'_L, $T'_R$are defined as follows:
  - The two functions of the upper subtables are equal to the functions of the upper subtables on the left. Namely, $h'_{L0} = h_{L0}$ and $h'_{R0} = h_{R0}$.
  - The two functions of the lower subtables are the *mirror image* of the functions of the lower subtables on the left. Namely, h'_{L1}, h'_{R1} are defined such that $h'_{L1} = h_{R1}$, and $h'_{R1} = h_{L1}$.

Bob's insertion algorithm. Bob needs to insert each of his items to one subtable in each of the tables T_L, T_R, T'_L, T'_R. He can do so by simply using Cuckoo hashing for each of these tables. For example, for the table T_L and its subtables T_{L0}, T_{L1}, Bob uses the functions h_{L0}, h_{L1} to insert each input x to either T_{L0} or T_{L1}. The same is applied to T_R, T'_L, and T'_R. In addition, Bob keeps a small stash of size $\omega(1)$ for each of the four tables. Overall, based on known properties of Cuckoo hashing, we can claim that the construction guarantees the following property:

Algorithm 1 (Mirror Cuckoo hashing)

1. Alice uses Cuckoo hashing to insert each item x to one of the subtables $T_{\mathrm{L}0}$, $T_{\mathrm{R}0}$, using the hash functions $h_{\mathrm{L}0}$, $h_{\mathrm{R}0}$.
2. Similarly, Alice uses Cuckoo hashing to insert each item x to one of the subtables $T_{\mathrm{L}1}$, $T_{\mathrm{R}1}$, using the hash functions $h_{\mathrm{L}1}$, $h_{\mathrm{R}1}$.
3. At this point, Alice observes the result of the first two steps. For some inputs x it happened that they were mapped to the same "column" in both of these steps. Namely, x was mapped to both $T_{\mathrm{L}0}$ and $T_{\mathrm{L}1}$, or to both $T_{\mathrm{R}0}$ and $T_{\mathrm{R}1}$. These are the "good" items, since they were mapped to the same column, as is required for all of Alice's inputs.
4. The other inputs of Alice, the "bad" items, were mapped to one column in Step 1 and to the other column in Step 2. Alice applies the following procedure to these items:
   (a) Each "bad" item x is removed from both locations to which it was mapped in Steps 1 and 2.
   (b) x is now inserted in either of $T'_{\mathrm{L}0}$, $T'_{\mathrm{R}0}$ using the hash functions $h'_{\mathrm{L}0} := h_{\mathrm{L}0}$, $h'_{\mathrm{R}0} := h_{\mathrm{R}0}$ with the same mapping as in Step 1.
   (c) x is also inserted in either of $T'_{\mathrm{L}1}$, $T'_{\mathrm{R}1}$ using the hash functions $h'_{\mathrm{L}1} := h_{\mathrm{R}1}$, $h'_{\mathrm{R}1} := h_{\mathrm{L}1}$ with the same mapping as in Step 2.

Claim. With all but negligible probability, it holds that for every input x of Bob, and for each of the four tables T_{L}, T_{R}, T'_{L}, T'_{R}, Bob inserts x to exactly one of the two subtables or to the stash.

Alice's insertion algorithm. Alice's operation is a little more complex and is described in Algorithm 1. Alice considers the two upper subtables on the left, $T_{\mathrm{L}0}$, $T_{\mathrm{R}0}$, as two subtables for standard Cuckoo hashing. Similarly, she considers the two lower subtables on the left, $T_{\mathrm{L}1}$, $T_{\mathrm{R}1}$, as two subtables for standard Cuckoo hashing. In other words, she considers the left top row and the left bottom row as standard Cuckoo hashing tables.

Alice then inserts each input item of hers to each of these two tables using standard Cuckoo hashing. (She also uses stashes of size $\omega(1)$ to store items which cannot be placed in the Cuckoo tables.) For some input items x it happens that x is inserted in the top row to $T_{\mathrm{L}0}$ and in the bottom row to $T_{\mathrm{L}1}$; or x is inserted in the top row to $T_{\mathrm{R}0}$ and in the bottom row to $T_{\mathrm{R}1}$. Therefore, x is inserted in two subtables in the same column. (x is denoted as "good" since this is the outcome that we want.)

Let x' be one of the other, "bad", items. Thus, x' is inserted in the top row to $T_{\mathrm{L}0}$ and in the bottom row to $T_{\mathrm{R}1}$, or vice versa. In this case, Alice removes x' from the tables on the left and inserts it to the tables T'_{L}, T'_{R} on the right. Since the hash functions that are used in T'_{L}, T'_{R} are equal to the functions used on the left side (where in the bottom row the functions are in reverse order), Alice does not need to run a Cuckoo hash insertion algorithm on the right side: Assume that x' was stored in locations $T_{\mathrm{L}0}[h_{\mathrm{L}0}(x')]$ and $T_{\mathrm{R}1}[h_{\mathrm{R}1}(x')]$ on the left. Then Alice inserts it to locations $T'_{\mathrm{L}0}[h'_{\mathrm{L}0}(x')] = T'_{\mathrm{L}0}[h_{\mathrm{L}0}(x')]$ and $T'_{\mathrm{L}1}[h'_{\mathrm{L}1}(x')] = T'_{\mathrm{L}1}[h_{\mathrm{R}1}(x')]$ on the right.

In other words, in a global view, one can see the algorithm as composed of the following steps: (1) First, all items are placed in the left tables. (2) Each subtable is divided in two copies, where one copy contains the good items and the other copy contains the bad items. (3) The subtable copies with the good items are kept on the left, whereas the copies with the bad items are moved to the right, where in the bottom row on the right we replace the order of the subtables.

This algorithm has two important properties: First, all items that were successfully inserted in the first step to the left tables will be placed in tables on either the left or the right hand sides. Moreover, each item will be placed in two subtables in the same column — the good items happened to initially be placed in this way in the left tables; whereas the bad items were in different columns on the left side but were moved to the same column on the right side. Hence, we can state the following claim:

Claim. With all but negligible probability, Alice inserts each of her inputs either to two locations in exactly one of T_{L}, T_{R}, T'_{L}, T'_{R} and to no locations in other tables, or to a stash.

Tables size. The total size of the tables is $8(1+\varepsilon)n$.

Stash size. With regards to stashes, each party needs to keep a stash for each of the Cuckoo hashing tables that it uses. Since Alice runs the Cuckoo hashing insertion algorithm only for the left tables and re-uses the mapping for the right tables, she needs only two stashes. Bob on the other hand runs the Cuckoo hashing insertion algorithm four times and hence needs four stashes. (In order to preserve simplicity, we omitted the stashes in Fig. 1 and Algorithm 1.) Given the result of [29], and our observation in the full version [40] about its applicability to non-constant stash sizes, it holds that a total stash of size $\omega(1)$ elements suffices to successfully map all items, except with negligible probability. We note that the size of the stash can be arbitrarily close to constant, e.g., it can be set to be $O(\log\log n)$ or $O(\log^* n)$. Essentially, for any function $f(n) \in \omega(n)$, the size of the stash can be $o(f(n))$.

4.2 Circuit-Based PSI from Mirror Cuckoo Hashing

Mirror Cuckoo hashing lets the parties map their inputs to tables of size $O(n)$ and stashes of size $\omega(1)$, with negligible failure probability. It is therefore straightforward to construct a PSI protocol based on this hashing scheme:

1. The parties agree on the parameters that define the size of the tables and the stash for mirror Cuckoo hashing. They also agree on the hash functions that will be used in each table.
2. Each party maps its items to the tables using the hash functions that were agreed upon.
3. The parties evaluate a circuit that performs the following operations:
   (a) For each bin in the tables, the circuit compares the item that Alice mapped to the bin to the item that Bob mapped to the same bin.

(b) Each item that Bob mapped to his stashes is compared with all items of Alice. Similarly, each item that Alice mapped to her stashes is compared with all items of Bob.

The properties of mirror Cuckoo hashing ensure: (1) If an item x is in the intersection, then there is exactly one comparison in which x is input by both Alice and Bob. (2) The number of comparisons in Step 3 is $\omega(n)$.

5 A Concretely Efficient Construction Through 2D Cuckoo Hashing

Two-dimensional Cuckoo hashing (a.k.a. 2D Cuckoo hashing) is a new construction with the following properties:

- It uses overall $O(n)$ memory (specifically, $8(1+\varepsilon)n$ in our construction, where we set $\varepsilon = 0.2$ in our experiments).
- Both, Alice and Bob, map each of their items to $O(1)$ memory locations (specifically, to two or four memory locations in our construction).
- If x appears in the input of both parties, then there is exactly one location to which both Alice and Bob map x.

The construction uses two tables, T_{L}, T_{R}, located on the left and the right side, respectively. Each of these tables is of size $4(1+\varepsilon)n$ and is composed of two smaller subtables: T_{L} is composed of the two smaller subtables T_{L0}, T_{L1}, while T_{R} is composed of the two smaller tables T_{R0}, T_{R1}. The hash functions h_{L0}, h_{L1}, h_{R0}, h_{R1} are used to map items to T_{L0}, T_{L1}, T_{R0}, T_{R1}, respectively. The tables are depicted in Fig. 2.

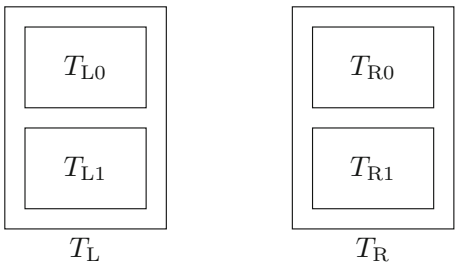


Fig. 2. The tables T_{L} and T_{R}, consisting of T_{L0}, T_{L1} and T_{R0}, T_{R1}, respectively.

Hashing is performed in the following way:

- Alice maps each of her items to all subtables on one of the two sides. Namely, each item x of Alice is either mapped to both bins $T_{\mathrm{L0}}[h_{\mathrm{L0}}(x)]$ and $T_{\mathrm{L1}}[h_{\mathrm{L1}}(x)]$ on the left side, or to bins $T_{\mathrm{R0}}[h_{\mathrm{R0}}(x)]$ and $T_{\mathrm{R1}}[h_{\mathrm{R1}}(x)]$ on the right side. In other words, ALICE maps each item to ALL subtables on one side.

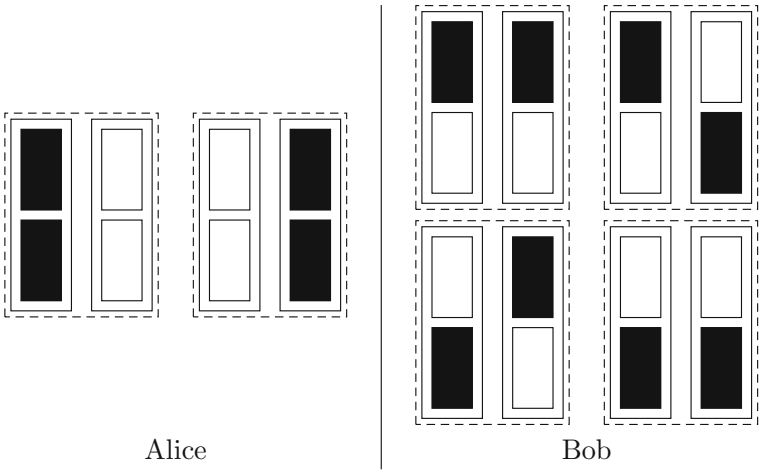


Fig. 3. The possible combinations of locations to which Alice and Bob map their inputs.

– Bob maps each of his items to one subtable on each side. This is done using standard Cuckoo hashing. Namely, each input x of Bob is mapped to one of the locations $T_{\mathrm{L}0}[h_{\mathrm{L}0}(x)]$ or $T_{\mathrm{L}1}[h_{\mathrm{L}1}(x)]$ on the left side, as well as mapped to one of the locations $T_{\mathrm{R}0}[h_{\mathrm{R}0}(x)]$ or $T_{\mathrm{R}1}[h_{\mathrm{R}1}(x)]$ on the right side. In other words, BOB maps each item to one subtable on BOTH sides.

The possible options for hashing an item x by both parties are depicted in Fig. 3. It is straightforward to see that if both parties have the same item x, there is exactly one table out of $T_{\mathrm{L}0}$, $T_{\mathrm{L}1}$, $T_{\mathrm{R}0}$, $T_{\mathrm{R}1}$ that is used by both Alice and Bob to store x.

We next describe a construction of 2D Cuckoo hashing, followed by a variant based on a heuristic optimization that stores two items in each table entry. The asymptotic behavior of the basic construction is analyzed in the full version [40]. In Sect. 6.1 we describe simulations for setting the parameters of the heuristic construction in order to reduce the hashing failure probability to below 2^{-40}.

5.1 Iterative 2D Cuckoo Hashing

This construction uses two tables, T_{L}, T_{R}, each of $4(1+\varepsilon)n$ entries. (In this construction, there is no need to assume that each table is composed of two subtables.) The parties associate two hash functions with each table, namely $h_{\mathrm{L}0}$, $h_{\mathrm{L}1}$ for T_{L}, and $h_{\mathrm{R}0}$, $h_{\mathrm{R}1}$ for T_{R}.

Bob uses Cuckoo hashing to insert each of his items into one location in each of the tables.

Alice inserts each item x either into the two locations $h_{\mathrm{L}0}(x)$ and $h_{\mathrm{L}1}(x)$ in T_{L}, or into the two locations $h_{\mathrm{R}0}(x)$ and $h_{\mathrm{R}1}(x)$ in T_{R}. This is achieved by Alice running a modified Cuckoo insertion algorithm that maps an item to two locations in one table, "kicks out" any item that is currently present in these locations and also removes the other occurrence of this item from the table, and then tries to insert this item into its two locations in the other table, and so on.

Algorithm 2 (Iterative 2D Cuckoo hashing)

1. Alice maps all of her items to table T_{L}, using simple hashing. That is, each item x is inserted in locations $h_{\mathrm{L}0}(x), h_{\mathrm{L}1}(x)$. Obviously, there will be entries in T_{L} that will have more than a single item mapped to them. Denote T_{L} as the active table.
2. For each entry in the active table with more than one item in it: remove all items – except for the item that was mapped to this entry most recently – and move them to the "relocation pool". For each of the removed items, remove the item also from its other appearance in the active table. (At the end of this step, all entries in the active table have at most one entry. However, there might be items in the relocation pool.)
3. If the relocation pool is empty, then stop (found a successful mapping).
4. Change the designation of the active table to point to the other table.
5. Move each item x from the relocation pool to locations $h_0(x), h_1(x)$ in the active table. (For example, if T_{R} is the active table, move x to $h_{\mathrm{R}0}(x), h_{\mathrm{R}1}(x)$.)
6. Go to Step 2.

Algorithm 3 (Iterative 2D Cuckoo hashing with bins of size 2)

The algorithm is identical to Algorithm 2, except for the following change in Step 2:

2. For each entry in the active table with more than *two* items in it: remove all items – except for the *two* items that were mapped to this entry most recently – and move them to the "relocation pool". For each of the removed items, remove the item also from its other appearance in the active table.

This is a new variant of Cuckoo hashing, where inserting an item into a table might result in four elements that need to be stored in the other table: storing x in $h_{\mathrm{L}0}(x), h_{\mathrm{L}1}(x)$ might remove two items, y_0, y_1, one from each location. These items are also removed from their other occurrences in T_{L}. They must now be stored in locations $h_{\mathrm{R}0}(y_0), h_{\mathrm{R}1}(y_0), h_{\mathrm{R}0}(y_1), h_{\mathrm{R}1}(y_1)$ in T_{R}.

It is not initially clear whether such a mapping is possible (with high probability, given random choices of the hash functions). We analyze the construction in the full version [40] and show that it only fails with probability $O(1/n)$. We ran extensive simulations, showing that the algorithm (when using a stash and a certain choice of parameters) fails with very small probability, smaller than 2^{-40}.

The insertion algorithm of Alice is described in Algorithm 2. The choice made in Step 2 of the algorithm, to first remove the oldest items that were mapped to the entry, is motivated by the intuition that it is more likely that the locations to which these items are mapped in the other table are free.

Storing two items per bin. It is known that the space utilization of Cuckoo hashing can be improved by storing more than one item per bin (cf. [15,37] or the review of multiple choice hashing in [47]). We take a similar approach and

use two tables of size $2(1+\varepsilon)n$ where each entry can store *two* items. (These tables have half as many entries as before, but each entry can store two items rather than one. The total size of the tables is therefore unchanged.) The change to the insertion algorithm is minimal and affects only Step 2. The new algorithm is defined in Algorithm 3.

Our experiments in Sect. 6.1 show that when using the same amount of space, then this variant of iterative 2D Cuckoo hashing performs better than the basic protocol with bins of size one. That is, it achieves a lower probability of hashing failure, namely of the need to use the stash, and requires less iterations to finish.

5.2 Circuit-Based PSI from 2D Cuckoo Hashing

This section describes how 2D Cuckoo hashing can be used for computing PSI. In addition, we describe two optimizations which substantially improve the efficiency of the protocol. The first optimization has the parties use permutation-based hashing [2] (as was done in [39]) in order to reduce the size of the items that are stored in each bin, and hence reduce the number of gates in the circuit. The second optimization is based on having each party use a single stash instead of using a separate stash for each Cuckoo hashing instance.

The PSI protocol is pretty straightforward given 2D Cuckoo hashing:

First, the parties agree on the hash functions to be used in each table. (These functions must be chosen at random, independently of the inputs, in order not to disclose any information about the inputs. Therefore, a participant cannot change the hash functions if some items cannot be mapped, and thus we seek parameter values that make the hashing failure probability negligible, e.g., smaller than 2^{-40}.)

Then, each party maps its items to bins using 2D Cuckoo hashing and the chosen hash functions. The important property is that if Alice and Bob have the same input item then there exists exactly one bin into which both parties map this item (or, alternatively, at least one of them places this item in a stash). Empty bins are padded with dummy elements. This ensures that no information is leaked by how empty the tables and stashes are.

Afterwards, the parties construct a circuit that compares, for each bin, the items that both parties stored in it. In addition, this circuit compares each item that Alice mapped to the stash with all of Bob's items, and vice versa. Since the number of bins is $O(n)$, the number of items in each bin is $O(1)$, and the number of items in the stash is $\omega(1)$, the total size of this circuit is $\omega(n)$. The parties can define another circuit that takes the output of this circuit and computes a desired function of it, e.g., the number of items in the intersection.

Finally, the parties run a generic MPC protocol that securely evaluates this circuit (cf. Sect. 6.3 for a concrete implementation and benchmarks).

Permutation-based Hashing. The protocol uses permutation-based hashing to reduce the bitlength of the elements that are stored in the bins and thus reduces the size of the circuit comparing them. This idea was introduced in [2]

and used for PSI in [39]. It is implemented in the following way. The hash function h that is used to map an item x to one of the β bins is constructed as follows: Let $x = x_L|x_R$ where $|x_L| = \log \beta$. We first assume that β is a power of 2 and then describe the general case. Let f be a random function with range $[0, \beta - 1]$. Then h maps an element x to bin $x_L \oplus f(x_R)$ and the value stored in the bin is x_R. The important property is that the stored value has a reduced bitlength of only $|x| - \log \beta$, yet there are no collisions (since if x, y are mapped to the same bin and store the same value, then $x_R = y_R$ and $x_L \oplus f(x_R) = y_L \oplus f(y_R)$ and therefore $x = y$).

In the general case, where β is not a power of two, the output of h is reduced modulo β and a stored extra bit indicates if the output was reduced or not.

For Cuckoo hashing the protocol uses two hash functions to map the elements to the bins in one table. To avoid collisions among the two hash functions, a stored extra bit indicates which hash function was used.

Using a Combined Stash. Recall that Alice uses 2D Cuckoo hashing, for which we show experimentally in Sect. 6.1 that no stash is needed. Bob, on the other hand, uses two invocations of standard Cuckoo hashing, and therefore when he does not succeed in mapping an item to a table, he must store it in a stash and compare it with all items of Alice. In this case, the parties cannot encode their items using permutation-based hashing, and therefore these comparisons must be of the full-length original values and not of the shorter values computed using permutation-based hashing as described before. Therefore, the size of the circuits that handle the stash values have a considerable effect on the total overhead of the protocol.

We observe that, instead of keeping several stashes, Bob can collect all the values that he did not manage to map to any of the tables in a *combined* stash. Suppose that he maps items to c tables and that we have an upper bound s which holds w.h.p. on the size of each stash. A naive approach would use c stashes of that size, resulting in a total stash size of $c \cdot s$. A better approach would be to use a single stash for all these items, since it is very unlikely that all stashes will be of maximal size, and therefore we can show that with the same probability, the size s' of the combined stash is much smaller than $c \cdot s$. To do so, we determine the upper bounds for the combined stash for $c = 2$: The probability of having a combined stash of size s' is $\sum_{i=0}^{s'} P(i) \cdot P(s' - i)$, where $P(i)$ denotes the probability of having a single stash of size i. The value of $P(i)$ is $O(n^{-i}) - O(n^{-(i+1)}) \approx O(n^{-i})$ [29]. We can estimate the exact values of these probabilities based on the experiments conducted by [39]: they performed 2^{30} Cuckoo hashing experiments for each $n \in \{2^{11}, 2^{12}, 2^{13}, 2^{14}\}$ and counted the required stash sizes. Using linear regression, we extrapolated the results for larger sets of 2^{16} and 2^{20} elements. Table 1 shows the required stash sizes when binding the probability to be below 2^{-40}: it turns out that for 2^{12} and 2^{16} elements the combined stash should include only one more element compared to the upper bound for a single stash, whereas for 2^{20} even the same stash size is sufficient.

Table 1. Stash sizes required for binding the error probability to be below 2^{-40} when inserting $n \in \{2^{12}, 2^{16}, 2^{20}\}$ elements into $2.4n$ bins using Cuckoo hashing.

Number of elements n	2^{12}	2^{16}	2^{20}
Single stash size s (from [39, Table 4])	6	4	3
Stash size for two separate stashes $s' = 2s$	12	8	6
Combined stash size s'	7	5	3

All in all, when comparing to the naive solution with two separate stashes, the combined stash size is reduced by almost a factor of 2x.

5.3 Extension to a Larger Number of Parties

Computing PSI between the inputs of more than two parties has received relatively little interest. (The challenge is to compute the intersection of the inputs of all parties, without disclosing information about the intersection of the inputs of any subset of the parties.) Specific protocols for this task were given, e.g., in [20,24,32]. We note that our 2D Cuckoo hashing can be generalized to m dimensions in order to obtain a circuit-based protocol for computing the intersection of the inputs of m parties. The caveat is that the number of tables grows to 2^m and therefore the solution is only relevant for a small number of parties.

We describe the case of three parties: The hashing will be to a set of eight tables $T_{x,y,z}$, where $x, y, z \in \{0, 1\}$. Any input item of P_1 is mapped to either all tables $T_{0,0,0}, T_{0,0,1}, T_{0,1,0}, T_{0,1,1}$, or to all tables $T_{1,0,0}, T_{1,0,1}, T_{1,1,0}, T_{1,1,1}$. Namely, the index x is set to either 0 or 1, and the input item is mapped to all tables with that value of x. Every input of P_2 is mapped either to all tables whose y index is 0, or to all tables where $y = 1$. Every input of P_3 is mapped either to all tables whose z index is 0, or to all tables where $z = 1$.

It is easy to see that regardless of the choices of the values of x, y, z, the sets of tables to which all parties map an item intersect in exactly one table. Therefore, the parties can evaluate a simple circuit that checks every bin for equality of the values that were mapped to it by the three parties. It is guaranteed that if the same value is in the input sets of all parties, then there is exactly one bin to which this value is mapped by all three parties. If some items are mapped to a stash by one of the parties, they must be compared with all items of the other parties, but the overhead of this comparison is $\omega(n)$ if the stash is of size $\omega(1)$.

The remaining issue is the required size of the tables. In the full version [40] we show that inserting an item into one of two (big) tables, such that the item is mapped to k locations in that table, requires tables of size greater than $k^2(1+\varepsilon)n$. When computing PSI between three parties using the method described above, we have eight (small) tables, where each party must insert its items to four tables in one plane or to four tables in the other plane. Each such set of four small tables corresponds to a big table in the analysis and is therefore of size $16(1 + \varepsilon)n$. The total size of the tables is therefore $32(1 + \varepsilon)n$.

5.4 No Extension to Security Against Malicious Adversaries

We currently do not see how to extend our hashing-based protocols to achieve security against malicious adversaries. As pointed out by [44], it is inherently hard to extend protocols based on Cuckoo hashing to obtain security against malicious adversaries. The reason is that the placement of items depends on the exact composition of the input set, and therefore a malicious party might learn the placement used by the other party.

Coming up with a similar argument as in [44], assume that in our construction in Fig. 3, Bob maps an item x to the two upper subtables and Alice maps x to the two left subtables. Now assume Alice maliciously deviates from the protocol and places x only in the upper left subtable, but not in the lower left one. This deviation may allow Alice to learn whether Bob placed x in the upper or lower subtables: For example, in a PSI-CA protocol Alice could use only dummy elements and x as an input set and if the cardinality turns out to be 1, then she knows that Bob placed x in the upper left subtable. However, the locations in which Bob places an item cannot be simulated in the ideal world as they depend on other items in his input set. Therefore, we see no trivial way to provide security against malicious adversaries based on 2D Cuckoo hashing.

6 Evaluation

This section describes extensive experiments that set the parameters for the hashing schemes, the resulting circuit sizes, and the results of experiments evaluating PSI using these circuits.

6.1 Simulations for Setting the Parameters of 2D Cuckoo Hashing

We experimented with the iterative 2D Cuckoo hashing scheme described in Sect. 5.1, set concrete sizes for the tables, and examined the failure probabilities of hashing to the tables.

Our implementation is written in C and available online at http://encrypto.de/code/2DCuckooHashing. It repeatedly inserts a set of random elements into two tables using random hash functions. The insertion algorithm is very simple: All elements are first inserted into the two locations to which they are mapped (by the hash functions) in the first table. Obviously, many table entries will contain multiple items. Afterwards, the implementation iteratively moves items between the tables, in order to reduce the maximum bin occupancy below a certain threshold (cf. Algorithms 2 and 3 in Sect. 5.1).

Run-time. We report in Sect. 6.3 the results of experiments analyzing the run-time of the 2D Cuckoo hashing insertion algorithm. Overall, the insertion time (a few milliseconds) is negligible compared to the run-time of the entire PSI protocol.

Hashing to bins of size 1. First, we checked if it is possible to use a maximum bin occupation of 1. For this, we set the sizes of each of the two tables to be $4.8n$ (corresponding to the threshold size of $4(1+\varepsilon)n$ in the analysis available in the full version [40], as well as twice the recommended size for Cuckoo hashing, since all elements are inserted twice). We ran the experiment 100 000 times with input size $n = 2^{12}$ and bitlength 32. For all except 828 executions it was possible to reduce the maximum bin occupation to 1 after at least 7 and at most 129 iterations of the insertion algorithm. On average, 20 iterations of the insertion algorithm were necessary to achieve the desired result. In said 828 cases there remained at least one bin with more than one item even after 500 iterations of the insertion algorithm. This implies that iterative 2D Cuckoo hashing works in principle, but, as standard Cuckoo hashing, requires a stash for storing the elements of overfull bins.

Hashing to bins of size 2. For PSI protocols it would be desirable to avoid having an additional stash on Alice's side. In standard Cuckoo hashing it is possible to achieve better memory utilization and less usage of the stash by using fewer bins, where each bin can store two items [47]. Therefore, we changed the parameters as follows: the table size is halved and reduced to $2.4n$, but each bin is allowed to contain two elements. This way, while consuming the same amount of memory as before, we try to achieve better utilization. We followed the paradigm that was described in Sect. 3.2 for the experimental analysis of the failure probability. Namely, we ran massive sets of experiments to measure the number of failures for several values of n and several table sizes, and given this data we (1) found confidence intervals for the failure probability for specific values of the parameters, and (2) found how the failure probability behaves as a function of n.

Our first experiment ran 2^{40} tests within ∼2 million core hours on the Lichtenberg[3] high performance computer of the TU Darmstadt for input size $n = 2^{12}$. We chose input size 2^{12} (instead of larger sizes like 2^{16} or 2^{20}) since running experiments with larger values of n would have taken even more time and would have simply been impractical. It turned out that the insertion algorithm was successful in reducing the maximum bin size to 2 (after at most 18 iterations) in all but one test.

Given this data, we calculated the confidence interval of the failure probability p. The probability of observing one failure in N experiments is $N \cdot p \cdot (1-p)^{N-1}$, where in our experiments $N = 2^{40}$. We checked the values of p for which the probability of this observation is greater than 0.001 and concluded that with 99.9% confidence, the failure probability for iterative 2D Cuckoo hashing with set size $n = 2^{12}$ and table size $2.4n$ lies within $\left[2^{-50}, 2^{-37}\right]$. (Namely, there is at most a 0.001 probability that we would have seen one failure in 2^{40} runs if p was greater than 2^{-37} or smaller than 2^{-50}.)

[3] See http://www.hhlr.tu-darmstadt.de/hhlr/index.en.jsp for details on the hardware configuration.

Measuring the dependence on the parameters. To get a better understanding on how the failure probability behaves for different input and table sizes, we performed a set of experiments that required another ∼3.5 million core hours. Concretely, we ran 2^{40} tests for each set size $n \in \{2^6, 2^8, 2^{10}\}$ and each table size in the range $2.2n$, $2.4n$, and $2.6n$. We also tested the table size $3.6n$ for $n \in \{2^6, 2^8\}$ as well as table sizes $3.0n$ and $3.2n$ for $n = 2^{10}$. The results for all experiments are given in Table 2 and are depicted in Fig. 4.

The results demonstrate that, w.r.t. the dependence on n, for set sizes $n \in \{2^6, 2^8, 2^{10}\}$ it can be observed that increasing the set size by factor 4x reduces the failure probability by factor 64x. (For larger set sizes, the number of failures

Table 2. Number of observed stashes for different table sizes and set sizes n when performing 2^{40} tests of iterative 2D Cuckoo hashing.

Table size	Stash size	$n = 2^6$	$n = 2^8$	$n = 2^{10}$	$n = 2^{12}$
$2.2n$	1	64 020	1 021	16	—
	2	154	1	0	—
	3	4	0	0	—
$2.4n$	1	31 033	499	8	1
	2	65	0	0	0
$2.6n$	1	16 014	270	5	—
	2	33	0	0	—
$3.0n$	1	—	—	0	—
$3.2n$	1	—	—	0	—
$3.6n$	1	1 202	17	—	—

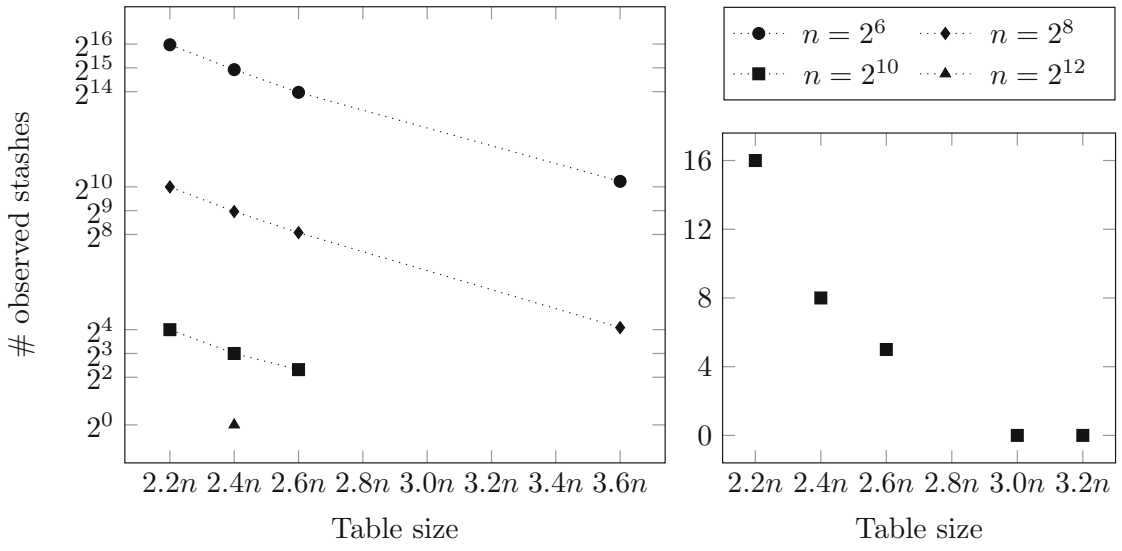


Fig. 4. Number of observed stashes for different table and set sizes when performing 2^{40} tests of iterative 2D Cuckoo hashing.

is too small to be meaningful.) These experiments also demonstrate that the dependence of the failure probability on n is $O(n^{-3})$. An intuitive theoretical explanation why the probability behaves this way is given in the full version [40]. As for the dependence on the table size, the failure probability decreases by a factor of 2x when increasing the table size in steps of $0.2n$ within the tested range $2.2n$ to $3.6n$.

From these results (a failure probability of at most 2^{-37} for $n = 2^{12}$ with table size $2.4n$ and a dependence of $O(n^{-3})$ of the failure probability on n) we conclude that the failure probability for $n \geq 2^{13}$ and table size $2.4n$ is at most 2^{-40}.

In total we spent about 5.5 million core hours on our experiments on the Lichtenberg high performance computer of the TU Darmstadt.

6.2 Circuit Complexities

We compare the complexities of the different circuit-based PSI constructions for two sets, each with n elements that have bitlength σ. We consider two possible bitlengths:

1. **Fixed bitlength:** Here, the elements have fixed bitlength $\sigma = 32$ bits (e.g., for IPv4 addresses).
2. **Arbitrary bitlength:** Here, the elements have arbitrary bitlength and are hashed to values of length $\sigma = 40 + 2\log_2(n) - 1$ bits, with a collision probability that is bounded by 2^{-40}. (See Appendix A of the full version of [41] for an analysis.) Therefore, we set the bitlength to $\sigma = 40 + 2\log_2(n) - 1$ bits.

For all protocols we report the circuit size where we count only the number of AND gates, since many secure computation protocols provide free computation of XOR gates. We compute the size of the circuits up to the step where single-bit wires indicate if a match was found for the respective element. We note that for many circuits computing functions of the intersection, this part of the circuit consumes the bulk of the total size. For example, computing the Hamming weight of these bits is equal to computing the cardinality of the intersection (PSI-CA). The size-optimal Hamming weight circuit of [5] has size $x - w_H(x)$ and depth $\log_2 x$, where x is the number of inputs and $w_H(\cdot)$ is the Hamming weight. The size of the Hamming weight circuit is negligible compared to the rest of the circuit. As another example, if the cardinality is compared with a threshold (yielding a PSI-CAT protocol), this only adds $3\log_2 n$ AND gates and depth $\log_2 \log_2 n$ using the depth-optimized construction described in [45], which is also negligible.

The size of the Sort-Compare-Shuffle circuit. The Sort-Compare-Shuffle circuit [26] has three phases. In the SORT phase, the two sorted lists of inputs are merged into one sorted list, which takes $2\sigma n \log_2(2n)$ AND gates. In the COMPARE phase, neighboring elements are compared to find the elements in the intersection, which takes $\sigma(3n - 1) - n$ AND gates. The SHUFFLE phase

randomly permutes these values and takes $\sigma(n\log_2(n) - n + 1)$ AND gates. To have a fair comparison with our protocols, we remove the SHUFFLE phase and let the COMPARE phase output only a single bit that indicates if a match was found for the respective element or not; this removes n multiplexers of σ-bit values from the COMPARE phase, i.e., σn AND gates. Hence, the total size is $2\sigma n\log_2(n) + 2\sigma n - n - \sigma + 2$ AND gates.

The size of the Circuit-Phasing circuit. The Circuit-Phasing circuit [39] has $2.4nm(\sigma - \log_2(2.4n) + 1) + sn(\sigma - 1)$ AND gates where m is the maximum occupancy of a bin for simple hashing and s is the size of the stash.

The size of our iterative 2D Cuckoo hashing construction of Sect. 5.2. Each of the following operations is performed twice for the left and right side: (1) For each of the $2.4n$ bins the shortened representation (cf. Sect. 5.2) of the single item in Bob's bin is compared with the two elements in the corresponding bin of Alice. (2) Bob has a stash of size s'. Each item in the stash is compared to all of Alice's items (using the full bitlength representation). Hence, the overall complexity is $4 \cdot 2.4n(\sigma - \log_2(2.4n) + 1) + s'n(\sigma - 1)$ AND gates, where s' is the size of the combined stash.

Concrete Circuit Sizes. The Sort-Compare-Shuffle construction [26] has a circuit of size $O(\sigma n \log n)$. The Circuit-Phasing construction [39] has circuit size $O(\sigma n \log n / \log\log n)$, while the asymptotic construction we present in this paper has a size of $\omega(\sigma n)$ and the iterative 2D Cuckoo hashing construction has an even smaller size.

For a comparison of the concrete circuit sizes, we use the parameters from the analysis in [39]: For $n = 2^{12}$ elements the maximum bin size for simple hashing is $m = 18$, for $n = 2^{16}$ we set $m = 19$, and for $n = 2^{20}$ we set $m = 20$. We set the stash size s and the combined stash size s' according to Table 1 (on page 21).

On the left side of Table 3 we compare the concrete circuit sizes for *fixed* bitlength $\sigma = 32$ bit. Our best protocol ("Ours Iterative Combined") improves over the best previous protocol by factor 2.0x for $n = 2^{12}$ (over [26]), by factor 2.7x for $n = 2^{16}$ (over [39]), and by factor 3.2x for $n = 2^{20}$ (over [39]).

On the right side of Table 3 we compare the concrete circuit sizes for *arbitrary* bitlength σ. Our best protocol (Ours Iterative Combined) improves over the best previous protocol by factor 1.8x for $n = 2^{12}$ (over [26]), by factor 2.8x for $n = 2^{16}$ (over [26]), and by factor 3.8x for $n = 2^{20}$ (over [39]).

Our constructions always have smaller circuits than both former constructions, and, due to our better asymptotic size, the savings become greater as n increases.

Circuit Depths. For some protocols, the circuit depth is a relevant metric (e.g., for the GMW protocol the depth determines the round complexity of the online phase). Our constructions have the same depth as the Circuit-Phasing protocol of [39], i.e., $\log_2 \sigma$. This is much more efficient than the depth of the

Table 3. Concrete circuit sizes in #ANDs for PSI variants on n elements of fixed bitlength $\sigma = 32$ (left) and arbitrary bitlength hashed to $\sigma = 40 + 2\log_2(n) - 1$ bits (right).

Protocol	Fixed bitlength $\sigma = 32$			Arbitrary bitlength		
	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{12}$	$n = 2^{16}$	$n = 2^{20}$
Sort-Compare-Shuffle [26]	3 403 746	71 237 602	1 408 237 538	6 705 091	158 138 299	3 478 126 515
Circuit-Phasing [39]	4 254 256	55 155 466	688 258 388	10 501 475	181 928 305	3 201 695 060
Separate stashes $s' = 2s$						
Ours iterative separate	2 299 801	26 153 770	313 183 300	5 042 482	71 137 681	1 081 999 223
Combined stash s' (cf. Table 1)						
Ours iterative combined	1 664 921	20 058 922	215 665 732	3 772 722	57 375 121	836 632 439

Sort-Compare-Shuffle circuit of [26] which is $O(\log \sigma \cdot \log n)$ when using depth-optimized comparison circuits.

Further Optimizations. So far, we computed the comparisons with a Boolean circuit consisting of 2-input gates: For elements of bitlength ℓ, the circuit XORs the elements and afterwards computes a tree of $\ell - 1$ non-XOR gates s.t. the final output is 1 if the elements are equal or 0 otherwise. This circuit allows to use an arbitrary secure computation protocol based on Boolean gates, e.g., Yao or GMW. The recent approach of [14] shows that for security against semi-honest adversaries the communication can be improved by using multi-input lookup tables (LUTs). Their best LUT has 7 inputs and requires only 372 bits of total communication (cf. [14, Table 4]). For computing equality, 6 of the non-XOR gates in the tree can be combined into one 7-input LUT. This improves communication of the Circuit-Phasing protocol of [39] and our protocols by factor $6 \cdot 256/372 = 4.1$x.

6.3 Performance

We empirically compare the performance of our iterative 2D Cuckoo hashing PSI-CAT protocol with a combined stash described in Sect. 5.2 with the Circuit-Phasing PSI-CAT protocol of [39]. As a baseline, we also compare with the public key-based PSI-CA protocol of [9,35,46] that leaks the cardinality to one party, and the currently best specialized PSI protocol of [31] that cannot be easily modified to compute variants of the set intersection functionality.

Implementation. Pinkas et al. [39] provide the implementation of their Circuit-Phasing PSI protocol as part of the ABY framework [13]. This framework allows to securely evaluate the PSI circuit using either Yao's garbled circuit or the GMW protocol, both implemented with most recent optimizations (cf. Sect. 2). However, since the evaluation in [39] showed that using the GMW protocol yields much better run-times, we focus only on GMW. ABY also implements the LUT-based evaluation of [14] (cf. Sect. 6.2), which we compare to GMW evaluation.

For the Circuit-Phasing PSI-CAT protocol, we extended the existing codebase with the Hamming weight circuit of [5] and the depth-optimized comparison circuit of [45] to compare the Hamming weight with a threshold. Based on this, we implemented our iterative 2D Cuckoo hashing PSI-CAT protocol by duplicating the code for simple hashing and Cuckoo hashing, combining the stashes, and implementing the iterative insertion algorithm. Our implementation is available online as part of the ABY framework at http://encrypto.de/code/ABY. For the DH/ECC-based protocol of Shamir/Meadows/De Cristofaro et al. [9,35,46], we use the ECC-based implementation of [39] available online at http://encrypto.de/code/PSI that already supports computing the cardinality (PSI-CA). The implementation of the special purpose BaRK-OPRF PSI protocol of [31] is taken from https://github.com/osu-crypto/BaRK-OPRF.

Benchmarking Environment. For our benchmarks we use two machines, each equipped with an Intel Core i7-4790 CPU @ 3.6 GHz and 16 GB of RAM. The CPUs support the AES-NI instruction set for fast AES evaluations. We distinguish two network settings: a LAN setting and a WAN setting. For the LAN setting, we restrict the bandwidth of the network interfaces to 1 Gbit/s and enforce a round-trip time of 1 ms. For the WAN setting, we limit the bandwidth to 100 Mbit/s and set a round-trip time of 100 ms. We instantiate all protocols corresponding to a computational security parameter of 128 bit and a statistical security parameter of 40 bit. All reported run-times are the average of 10 executions with less than 10% variance.

Benchmarking Results. In Table 4, we give the run-times for $n \in \{2^{12}, 2^{16}, 2^{20}\}$ elements[4] of bitlength $\sigma = 32$ (suitable, e.g., for IPv4 addresses). The corresponding communication is given in Table 6. We do not use the LUT-based evaluation in the LAN setting since there is little need for better communication while the run-times are not competitive. However, to demonstrate the advantages of the LUT-based evaluation in the WAN setting, we compare the protocols when running with a single thread and four threads.[5]

Run-times (Tables 4 and 5). In comparison with the Circuit-Phasing PSI-CAT protocol of [39] in Table 4, our iterative combined PSI-CAT protocol is faster by factor 1.4x for $n = 2^{12}$ and up to factor 2.8x for $n = 2^{20}$. This holds when the circuit is evaluated with GMW in both network settings and for both 1 and 4 threads. With LUT-based evaluation [14], we observe a further improvement for the circuit-based protocols by about 13% in the WAN setting, but only for medium set sizes of $n = 2^{16}$ and 4 threads due to the higher computation complexity.

The circuit-based protocols have two steps: mapping the input items to the tables, and securely evaluating the circuit. The run-times of the hashing step are

[4] Unfortunately, the LUT-based implementation of [14] was not capable of evaluating the PSI circuits for $n = 2^{20}$ elements.

[5] We do not provide benchmarks with multiple threads for the DH/ECC PSI-CA protocol since the implementation of [39] does not support multi-threading.

shown in Table 5. The times for Cuckoo hashing into two tables in our PSI-CAT protocol are exactly twice of those for Cuckoo hashing into one table in [39]. Compared to simple hashing, our 2D Cuckoo hashing is slower by factor 1.6x up to factor 2.1x due to the additional iterations. However, all in all, the hashing procedures are by 2–3 orders of magnitude faster than the times for securely evaluating the circuit, and therefore negligible w.r.t. the overall run-time.

In comparison with the DH-based PSI-CA protocol of [9,35,46], our iterative combined PSI-CAT protocol is faster by factor 1.5x for $n = 2^{12}$ up to factor 91x for $n = 2^{20}$ in the LAN setting with a single thread. Also in the WAN setting with a single thread, our protocol is faster (except for small sets with $n = 2^{12}$), despite the substantially lower communication of the DH-based protocol described below. In both network settings even the best measured run-times of our PSI-CAT protocol are between 19x to 36x slower than the BaRK-OPRF specialized PSI protocol of [31], but our protocols are generic.

Communication (Table 6). The communication given in Table 6 is measured on the network interface, so these numbers are slightly larger than the theoretical communication (derived from the number of AND gates on the left side in Table 3) due to TCP/IP headers and padding of messages. The lowest communication is

Table 4. Total run-times in ms for PSI variants on n elements of bitlength $\sigma = 32$ bit.

Protocol	Network setting	LAN			WAN				
	Circuit evaluation protocol	GMW [21]			GMW [21]			LUT [14]	
	Set size n	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}
DH/ECC PSI-CA [9,35,46]		3 296	49 010	7 904 054	4 082	51 866	8 008 771	4 082	51 866
BaRK-OPRF PSI [31]		113	295	3 882	540	1 247	14 604	540	1 247
1 Thread									
Circuit-Phasing PSI-CAT [39]		3 170	20 401	242 235	15 143	99 433	1 042 712	19 951	117 438
Ours iterative separate PSI-CAT		2 433	11 251	122 008	11 210	57 474	547 950	15 656	70 545
Ours iterative combined PSI-CAT		2 220	9 076	86 648	10 060	45 252	389 891	12 999	56 179
4 Threads									
Circuit-Phasing PSI-CAT [39]		2 333	10 600	123 765	12 492	97 480	987 459	15 471	76 184
Ours iterative separate PSI-CAT		1 903	6 273	64 324	9 361	56 141	541 677	11 946	46 797
Ours iterative combined PSI-CAT		1 694	5 177	49 417	8 793	44 596	376 591	9 413	39 272

Table 5. Run-times in ms for hashing n elements of bitlength $\sigma = 32$ bit.

Hashing procedure	Set size n	2^{12}	2^{16}	2^{20}
Circuit-Phasing PSI-CAT [39]				
Simple hashing		3.50	27.96	557.54
Cuckoo hashing		2.43	15.87	391.16
Ours iterative PSI-CAT				
2D Cuckoo hashing		6.23	58.90	873.19
Cuckoo hashing (for two tables with a combined stash)		4.85	31.75	782.32

Table 6. Communication in MB for PSI variants on n elements of bitlength $\sigma = 32$ bit.

Protocol \ Set size n	2^{12}	2^{16}	2^{20}
DH/ECC PSI-CA [9,35,46]	0.4	6.6	106.0
BaRK-OPRF PSI [31]	0.53	8.06	127.20
GMW [21]			
Circuit-Phasing PSI-CAT [39]	121.9	1 588.9	20 028.5
Ours iterative separate PSI-CAT	72.3	826.1	9 971.4
Ours iterative combined PSI-CAT	52.7	638.8	6 950.6
LUT [14]			
Circuit-Phasing PSI-CAT [39]	32.6	418.1	—
Ours iterative separate PSI-CAT	19.4	221.3	—
Ours iterative combined PSI-CAT	14.3	171.3	—

achieved by the DH-based PSI-CA protocol of [9,35,46] which is in line with the experiments in [39]. Our best protocol for PSI-CAT has between 132x (for $n = 2^{12}$) and 66x (for $n = 2^{20}$) more communication than the DH-based PSI-CA protocol when evaluated with GMW. Recall, however, that our protocol does not leak the cardinality. Our best protocol improves the communication over the PSI-CAT protocol of [39] by factor 2.3x (for $n = 2^{12}$) to 2.9x (for $n = 2^{20}$). When using LUT-based evaluation of [14], we observe that the communication of all circuit-based PSI-CAT protocols improves over GMW by factor 3.7x which is close to the theoretical upper bound of 4.1x (cf. Sect. 6.2). Still, our best LUT-based protocol has more than 20x higher communication than the BaRK-OPRF specialized PSI protocol of [31], but it is generic.

Application to privacy-preserving ridesharing. Our PSI-CAT protocol can easily be extended for the privacy-preserving ridesharing functionality of [23], where the intersection is revealed only if the size of the intersection is larger than a threshold. The authors of [23] give a protocol that securely computes this functionality, but has quadratic computation complexity. By slightly extending our circuit for PSI-CAT to encapsulate a key that is released only if the size of the intersection is larger than the threshold and using this key to symmetrically encrypt the last message in any linear complexity PSI protocol (e.g., [31,39,41,42]), we get a protocol with almost linear complexity. Our key encapsulation would take less than 3 s for $n = 2^{12}$ elements (cf. our results for PSI-CAT in Table 4), whereas the solution of [23] takes 5 627 s, i.e., we improve by factor 1 876x and also asymptotically.

Acknowledgments. We thank Oleksandr Tkachenko for his invaluable help with the implementation and benchmarking. We also thank Moni Naor for suggesting the application to achieve differential privacy. This work has been co-funded by the DFG as part of project E4 within the CRC 1119 CROSSING and by the German Federal

Ministry of Education and Research (BMBF), the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, and the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

References

1. Amossen, R.R., Pagh, R.: A new data layout for set intersection on GPUs. In: International Symposium on Parallel and Distributed Processing (IPDPS) (2011)
2. Arbitman, Y., Naor, M., Segev, G.: Backyard cuckoo hashing: constant worst-case operations with a succinct representation. In: FOCS (2010)
3. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: CCS (2013)
4. Asokan, N., Dmitrienko, A., Nagy, M., Reshetova, E., Sadeghi, A.-R., Schneider, T., Stelle, S.: CrowdShare: secure mobile resource sharing. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 2013. LNCS, vol. 7954, pp. 432–440. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38980-1_27
5. Boyar, J., Peralta, R.: Concrete multiplicative complexity of symmetric functions. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 179–189. Springer, Heidelberg (2006). https://doi.org/10.1007/11821069_16
6. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS (2017)
7. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 125–142. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01957-9_8
8. Davidson, A., Cid, C.: An efficient toolkit for computing private set operations. In: Pieprzyk, J., Suriadi, S. (eds.) ACISP 2017. LNCS, vol. 10343, pp. 261–278. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59870-3_15
9. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 218–231. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35404-5_17
10. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_13
11. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_13
12. Debnath, S.K., Dutta, R.: Secure and efficient private set intersection cardinality using bloom filter. In: Lopez, J., Mitchell, C.J. (eds.) ISC 2015. LNCS, vol. 9290, pp. 209–226. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23318-5_12
13. Demmler, D., Schneider, T., Zohner, M.: ABY – a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
14. Dessouky, G., Koushanfar, F., Sadeghi, A.-R., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the communication barrier in secure computation using lookup tables. In: NDSS (2017)

15. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. Theoret. Comput. Sci. **380**(1–2), 47–68 (2007)
16. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: CCS (2013)
17. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006). https://doi.org/10.1007/11787006_1
18. Eppstein, D., Goodrich, M., Mitzenmacher, M., Torres, M.: 2–3 cuckoo filters for faster triangle listing and set intersection. In: Symposium on Principles of Database Systems (PODS) (2017)
19. Freedman, M.J., Hazay, C., Nissim, K., Pinkas, B.: Efficient set intersection with simulation-based security. J. Cryptol. **29**(1), 115–155 (2016)
20. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_1
21. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC (1987)
22. Gonnet, G.H.: Expected length of the longest probe sequence in hash code searching. J. ACM **28**(2), 289–304 (1981)
23. Hallgren, P., Orlandi, C., Sabelfeld, A.: PrivatePool: privacy-preserving ridesharing. In: Computer Security Foundations Symposium (CSF) (2017)
24. Hazay, C., Venkitasubramaniam, M.: Scalable multi-party private set-intersection. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10174, pp. 175–203. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_8
25. Huang, Y., Chapman, P., Evans, D.: Privacy-preserving applications on smartphones. In: Hot Topics in Security (HotSec) (2011)
26. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
27. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security (2011)
28. Ion, M., Kreuter, B., Nergiz, E., Patel, S., Saxena, S., Seth, K., Shanahan, D., Yung, M.: Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738 (2017)
29. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: cuckoo hashing with a stash. SIAM J. Comput. **39**(4), 1543–1561 (2009)
30. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. In: PoPETs, vol. 2017(4) (2017)
31. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS (2016)
32. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multiparty private set intersection from symmetric-key techniques. In: CCS (2017)
33. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
34. Kreuter, B.: Secure multiparty computation at Google. In: Real World Crypto Conference (RWC) (2017)
35. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: S&P (1986)

36. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: European Symposium on Algorithms (ESA) (2001)
37. Panigrahy, R.: Efficient hashing with lookups in two memory accesses. In: ACM-SIAM Symposium on Discrete Algorithms (SODA) (2005)
38. Pettai, M., Laud, P.: Combining differential privacy and secure multiparty computation. In: ACSAC (2015)
39. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: USENIX Security (2015)
40. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: Cryptology ePrint Archive, Report 2018/120 (2018)
41. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX Security (2014)
42. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. (TOPS) **21**(2) (2018)
43. Rindal, P., Rosulek, M.: Improved private set intersection against malicious adversaries. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 235–259. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_9
44. Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: CCS (2017)
45. Schneider, T., Zohner, M.: GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 275–292. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_23
46. Shamir, A.: On the power of commutativity in cryptography. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 582–595. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_100
47. Wieder, U.: Hashing, load balancing and multiple choice. Found. Trends Theoret. Comput. Sci. **12**(3–4), 275–379 (2017)
48. Yao, A.C.: How to generate and exchange secrets. In: FOCS (1986)
49. Yung, M.: From mental poker to core business: why and how to deploy secure computation protocols? In: CCS (2015)
50. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole: reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8